Extraction Of Arabic Word Roots

An Approach Based on Computational Model and Multi-Backpropagation Neural Networks

HASAN MUAIDI AL-SERHAN

MSc, Yarmouk University – JORDAN, 2003

BSc, Yarmouk University – JORDAN, 1987

PhD

July, 2008

Extraction Of Arabic Word Roots

An Approach Based on Computational Model and Multi-Backpropagation Neural Networks

BY

HASAN MUAIDI AL-SERHAN

MSc, Yarmouk University – JORDAN, 2003

BSc, Yarmouk University – JORDAN, 1987

A thesis submitted in partial fulfillment
of the requirements for the Degree of
Doctor of Philosophy
in Computer Science

School of Computing
Faculty of Computing Sciences and Engineering

dmu.ac.uk
DE MONTFORT UNIVERSITY
LEICESTER

DE MONTFORT UNIVERSITY
LEICESTER - U.K.

July, 2008

Dedication

To The Soul Of My Father, whom I fondly remember and with love and affection dedicate my system to immortalise his name.

To My Mother, who gave me endless valuable advice and help. Her unconditional love, prayer and encouragement made it easier for me to achieve my goals.

To My Lovely Wife, who gave me the love and support throughout my life that has allowed me to be who I am today. Your endurance, sacrifice and patience has been a gift that I will cherish always. May ALLAH guide our next steps together along our journey of life.

To My Kids: Laith, Hadeel, Ghaith, Aseel and Hala, those always kept my spirits high. The life without them means nothing.

Acknowledgments

First and foremost, I thank ALLAH for his blessings and bounties he gave to me. Secondly, I would like to express my considerable gratitude to the following institutions and people who, in one way or another, provided me with the possibility to conduct the research presented here.

- The best thanks go to the Al-Balqa Applied University (BAU) - JORDAN and its president for providing me with a scholarship to pursue my PhD degree.

- The grateful thanks go to my first supervisor Dr. Aladdin Ayesh, for his invaluable advice, guidance and support throughout and during the course of my study, also for his constant encouragement to publish my work. He stands behind the completion of this thesis and makes it accessible.

- The deepest thanks and gratitude to my second supervisor Prof. Robert John for his critical comments, professional guidance, and insightful suggestions to finalise this thesis, also many thanks to my third supervisor Dr. John Cowell for his encouragement and support.

- The grateful thanks go to Fahed Ashour[1] for his invaluable linguistic consultation in this thesis.

[1]Fahed Ashour is an Arabic linguist [39][40][38], email: fahed.ashour@gsaa.uni-halle.de. Martin Luther University, Halle, Germany.

- I would also like to thank the members of my family in which I grew up especially my mother, brothers, and sisters without whom none of this would exist. And above all, I thank the members of my family with whom I share the daily life; your love and patience have been the basic requisite for this work.

- I would also like to convey my sincere gratitude to my closest friend Shihadeh Al-Qrainy for the special days that we have spent together during our long trip.

- Special thanks also go to all of my friends in Leicester for the nice days that we spent together.

- I would also specially like to thank all members of the computational intelligence Centre Research Group at De Montfort University, and all my friends for their support and valuable guidance while completing my thesis.

Hasan Muaidi Al-Serhan

LEICESTER - U.K.

Abstract

Stemming is a process of extracting the root of a given word, by stripping off the affixes attached to this word. Many attempts have been made to address the stemming of Arabic words problem. The majority of the existing Arabic stemming algorithms require a complete set of morphological rules and large vocabulary lookup tables. Furthermore, many of them give more than one potential stem or root for a given Arabic word. According to Ahmad [11], the Arabic stemming process based on the language morphological rules is still a very difficult task due to the nature of the language itself.

The limitations of the current Arabic stemming methods have motivated this research in which we investigate a novel approach to extract the word roots of Arabic language named here as MUAIDI-STEMMER [2]. This approach attempts to exploit numerical relations between Arabic letters, avoiding having a list of the root and pattern of each word in the language, and giving one root solution. This approach is composed of two phases. **Phase I** depends on a basic calculations extracted from linguistic analysis of Arabic patterns and affixes. **Phase II** is based on artificial neural network trained by backpropagation learning rule. In this proposed phase, we formulate the root extraction problem as a classification problem and the neural network as a classifier tool. This study demonstrates that a neural network can be effectively used to ex-

[2] **Muaidi** is the author father's name.

tract the word roots of Arabic language

The stemmer developed is tested using **46,895** Arabic word types[3]. Error counting accuracy evaluation was employed to evaluate the performance of the stemmer. It was successful in producing the stems of **44,107** Arabic words from the given test datasets with accuracy of **94.81%**.

[3]Types mean distinct or unique words.

List of Publications

The following is a summary of the publications were derived from this research:

1. Al-Serhan, H. M. and A. S. Ayesh (2006). **"A Triliteral Word Roots Extraction Using Neural Network for Arabic"**. IEEE International Conference on Computer Engineering and Systems (ICCES'06), 436-439, Cairo, EGYPT.

2. Al-Serhan, H. M. and A. S. Ayesh (2006). **"An Application of Neural Network for Extracting Arabic Word Roots"**. WSEAS TRANSACTIONS on COMPUTERS 5(11): 2623-2627.

3. Al-Serhan, H. M. and A. S. Ayesh (2007). **"Extraction Arabic Word Roots Using Neural Network"**. In the 2nd Jordan International Conference on Computer Science & Engineering (JICCS'E06), 112-116, Amman, JORDAN.

4. Al-Serhan, H. M. and A. S. Ayesh (2008). **"Extraction Arabic Word Roots Using Multi-Backpropagation Neural Networks"**. Journal of Natural Language Engineering, Cambridge University Press, U.K. (Submitted after first revision).

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Overview

This chapter presents a brief description of the contents and the structure of this thesis as well as a slightly more lengthy discussion on the problem of extracting the Arabic word roots. The presentation covers a small background, the problem statement, the research questions and the objectives of the study, and the scope and limitation of the study. The last part of this chapter presents the outline of the thesis, and gives an overview of each chapter.

1.2 Background

The study described in this thesis addresses a problem of extracting the roots of Arabic words. The problem belongs to the area of computational linguistics. Computational linguistics is a field of artificial intelligence dealing with the logical modeling of natural language from a computational perspective [103]. It unites two areas that are quite different in appearance, computer science and natural languages. The following is a summarisation of the areas that may regarded as properly included within the discipline of computational linguistics [103]:

- Design of corpus linguistics,
- Design of parsers for natural languages,
- Design of taggers,
- Design of machine translation,
- Design of morphological analysers (e.g., Stemmers).

This study falls in the last area (morphological analysers), the rest of this section introduces a brief background.

In the last few years there has been an enormous growth in the amount of text documents available in the Internet and its recent popularity, particularly the World Wide Web (WWW), has accelerated its growth. Concomitant with this popularity has come an exponential-like growth in the volume of information available, and the emphasis on accessing the Internet focuses on finding the relevant information as quickly as possible [106].

This rapid increase in the volume of electronic information highlights the need for a new breed of search and information retrieval tools to reduce the time spent searching or browsing for information in the WWW.

The science of information retrieval nowadays can be seen as one of the most important technologies that help us to find all textual information we need within a structure that we can search and analyse. One of the attempts that improved the performance of the information retrieval systems is the usage of what is so-called word stemming [106]. **Stemming** is a computational process by which the words are reduced to their base forms (roots). This is usually done by stripping the word affixes.

1.3 General Problem Statement

One of the challenges facing the developers of Arabic text retrieval systems is the absence of a strong and an effective stemming algorithm. Arabic is

morphologically a complex language [111], it uses both kinds of morphologies: inflectional and derivational morphologies. As a result of these types of morphology, a single word may yield hundreds or even thousands of variant forms [34]. So, the morphology of Arabic can be viewed as a tool that enables the language to grow and develop [133]. As a result, the richness of word form variations can have a strong impact on the effectiveness of information retrieval systems [34].

Many studies showed that using roots as an index word in information retrieval systems gave much better results than using full words [134][96][54] [52][35]. Several studies in Arabic language [18][6][80] suggested that word stemming and root searching as techniques of information retrieval are the most effective ways. For example, Al-Kharashi and Evens [18] show the effectiveness of root-based retrieval method over the word-based method. The root-based method increased the number of documents retrieved in Arabic language by between **35** and **60** times. According to Larkey et al. [96], stemming has a positive effect in the highly inflected language such as Arabic language.

In addition to information retrieval systems, stemming can be used in many Arabic natural language processing applications [107]. It can be used in text generation to generate different parts-of-speech of a given word. Also stemming can be used in datamining, data compression, and spell checking applications, some details of such applications are presented in Chapter 3. In spite of the rapid research conducted in other languages, Arabic language still suffers from the shortages of researchers and development.

As mentioned before, Arabic language has a rich and complex morphology [101]. In many cases, one orthographic word consists of many semantic and syntactic words [92]. In English language a word is a single entity, while in Arabic language a single word could be a complete sentence (see Section 2.5 of Chapter 2). The word formation process in Arabic depends mainly on roots, patterns and affixes. A root is a sequence of three or more consonants. A pattern is a sequence of consonants, vowels, and slots for the root letters. Unlike

English language, the affixes in Arabic are very complex. They include prefixes, suffixes, and infixes. Prefixes are attached at the beginning of a word, suffixes are attached at the end, while infixes are inserted in the middle of a word. Infixes make the morphological analysis of the Arabic language a hard process [133].

Arabic is a structured language [133], in which the word formation process is usually done through the interdigitation of the root letters with a vocalic pattern in the root slots, and then might be accompanied by a set of affixes. An Arabic word could have repeated forms and combinations of these affixes, i.e., more than one affix may be attached to a word. The reverse of the word formation process is the so-called **root extraction**. To extract a root for a given word, all the affixes should be removed; it is not an easy problem even for humans.

Many attempts have been made to address the problem of root extraction. The evaluation of these techniques will be presented in Chapter 3. The majority of the existing Arabic stemming algorithms use a large set of rules, and many of them also refer to a lookup table of patterns and roots. In consequence, this requires a large storage space to store these tables, and time to access the information. Due to the above, there is no standard root-stemming algorithm for Arabic language. In a recent work done in 2007, Momani et al.[105] stated:

> "Arabic as a strong morphological language suffers from the unavailability of a standard Arabic root extraction algorithm" ([105], Page 1).

The limitations of the current Arabic stemming methods have motivated this author to investigate a novel approach to be used in the extraction of the word roots of Arabic language. This approach attempts to exploit the numerical relations between Arabic letters, avoiding having a list of the root and pattern of each word in the language. Furthermore, this approach has no use of the set of linguistic rules.

In this study, a computational approach to the problem of extracting the roots of Arabic words is presented. This approach is in one part depend on a basic calculation extracted from the analysis of Arabic roots and patterns whereas, in the other part it is use a multi-backpropagation neural network [76].

In this study, the reason behind choosing the neural networks is that in the recent years they have become a popular tool for research in the computational linguistics field. Neural networks have many advantages over the other approaches. One of these advantages is that neural networks can be efficient when the linguistic and the grammar rules are not known due to the complexity of the morphology of the language itself or when no human expert is available. In these cases, if there are sufficient training data, the neural networks can be able to learn from the given data at a comparable level to human experts.

The significance of the present work lies in the fact that this is being the first effort to solve the problem of the root-stemming Arabic words using (1) a computational approach exploiting numerical relations between Arabic letters, (2) using a neural networks approach.

This study is anticipated that will be helpful in the development of any future information retrieval systems for Arabic language. Also, it is hoped that this research will open a new window or a new view in processing many issues facing the development of Arabic natural language processing applications rather than using conventional techniques. If enough research is done, in the author's view, artificial neural networks will become one of the most popular methods dealing with such applications. The developed solution method is discussed at length in Chapters 5 and 6.

1.4 Research Question and Objectives

As mentioned before, stemming algorithms can be used in many Arabic natural language processing applications, and there are many attempts have been

made to address the development such algorithms using different approaches. Most of the existing Arabic stemming algorithms require a complete set of morphological rules and large vocabulary lookup tables. Furthermore, many of them give more than one potential stem or root for a given Arabic word. According to Ahmad [11], the Arabic stemming process based on the language morphological rules is still a very difficult task due to the nature of the language itself. Thus, the research questions of this study can be summarised as:

1. **Can we identify and exploit numerical relations between letters for Arabic root extraction ?.**

2. **Can we use the artificial neural networks as a tool to extract the roots of Arabic words ?.**

So, the overall goal of this study is to develop a root-stemming algorithm for Arabic language giving only one root solution. This is done by investigating the viability of applying a computational and a neural networks approach, avoiding having a pattern and an affix lists and a set of linguistics rules. The following specific objectives have been set to achieve the main goal:

- A comprehensive literature review of stemming algorithms and morphological analysers that have been developed for Arabic language. This provides a summarisation of what has been done in the literature of this field.

- Review different stemming algorithms that have been developed for English language.

- Analyse the Arabic roots based on four famous Arabic dictionaries, and generate a one common dictionary that combines all the roots in these dictionaries.

- Compile and analyse a list of pattern forms in Arabic language.

- Compile a list of affix letters used in Arabic language.

- Analyse the frequency of the affix letters in Arabic language.

- Compile a list of stopwords in Arabic language.
- Divide a complex problem (extraction the Arabic word roots) into simpler sub-tasks.
- Generate a neural network modules to work on these sub-tasks.
- Select a suitable Arabic corpus and prepare it for processing.
- Buildup an annotated corpus for the Arabic language containing Arabic words and their morphological features.
- Develop a computer program for the presented root-stemming algorithm.
- Test and evaluate the developed algorithm using a test data in order to check its performance.

1.5 Significant Research Contributions

This research provides a novel contributions to the field of the Arabic computational linguistics which are summarised as follows.

1- A Linguistic Analysis of Arabic Patterns and Affixes

In order to study the numerical relations between the Arabic letters, the author successfully surveyed **1,893** conceivable pattern forms based on the trilliteral roots. The analysis of surveyed patterns lead to generate a binary score matrices that show the possibility of the letter to be an affix letter. Furthermore, a comprehensive lists of Arabic prefixes, infixes, and suffixes are extracted from these patterns. We extracted **204** prefixes, **7** infixes and **193** suffixes. Also, the frequency of the affix letters in Arabic language is anlysed.

2- The Stemming Algorithm of Arabic language

The main contribution of this thesis is to develop a stemming algorithm to extract the roots of Arabic words. This algorithm is called **MUAIDI-STEMMER** [1].

[1] **Muaidi** is the author father's name.

The developed stemmer is a computational approach tries to exploit the numerical relations between letters that are identified by Arabic linguists to be affixes. The MUAIDI-STEMMER is conducted in two phases: Phase I and Phase II. **Phase I** is a computational module based on the analysis of Arabic patterns and the affix letters (Contribution 1), while **Phase II** is based on artificial neural networks trained in which we formulated the root extraction problem as a classification problem and the neural network as a classifier.

3- An Annotated Arabic Corpus

The existence of an annotated corpus is essential in the development of many natural language processing applications. Furthermore, a good sized corpus can show a significant language morphological behavior. For the purpose of this thesis, an annotated Arabic corpus is compiled and designed in which every word in this corpus has assigned its morphological features. The main source of the annotated corpus is extracted from Latifa Al-Sulaiti Arabic corpus [29][30] (**Corpus of Contemporary Arabic**). In this research, the annotated corpus is used for training and the testing the developed neural networks (Chapters 6 and 7). The steps of the generation the Annotated Arabic corpus is presented in Chapter 6.

4- A Compiled Arabic Dictionary

The extraction of the Arabic roots is the main aim of this thesis. So, the availability of a reliable dictionary of Arabic roots is considered as an essential component of the developing of any Arabic stemmer. Such this comprehensive dictionary does not exist as an electronic form. Therefore **27,457** roots were manually collected from four famous Arabic dictionaries. These roots are grouped in one dictionary called the **common dictionary**. The total number of distinct roots in this dictionary is **12,619**. The common dictionary is used in Chapter 6 during the building-up of the annotated corpus to check the validation of the roots. Also, it is used in Chapter 7 during the evaluation process for the same reason.

1.6 Scope and Limitation of the Study

This study concerns only Arabic language with no attention to other languages. This is due to the fact that every language has its own morphological features and its large number of rules.

The study focuses only on the triliteral roots (three letters) of Arabic language. Since, the overwhelming majority of roots in Arabic language are triliteral [127][25]. Al-Shalabi [89] states that **95%** of the Arabic roots are triliteral roots. Furtheremore, in the study of Ali [36], the Quran, the holy book of Islam, consists of **77,888** words, and it has only **1,633** trilileral roots **(97%)**, and only **45** non-trilileral roots **(03%)**. So the other root lengths are beyond the scope of this study.

1.7 Thesis Structure

This thesis is organised into eight chapters and eight appendices that cover the issues and topics of this study. The following descriptions give brief overview of the chapters to follow.

CHAPTER 2 This chapter deals with Arabic language structure and syntax which is an essential background to this study. It starts by describing the features of Arabic writing system. Then it discusses the derivational and inflectional morphology of the Arabic language. Finally, it shows the classification of Arabic words.

CHAPTER 3 This chapter deals with the stemming algorithms. It starts by defining the stemming concept. Then it describes the usage of stemming and its importance in natural language processing applications. The classifications of stemming approaches for English and Arabic languages are presented in this chapter. Then the chapter illustrates the previously

developed stemmers for English and Arabic languages. Finally, the chapter presents a brief introduction to artificial neural networks. This is given for the readers who have no previous background in artificial neural networks.

CHAPTER 4 This chapter covers the linguistic analysis of Arabic language: roots, patterns and affixes. It starts by highlighting the definition of a term root. Then it shows the statistical analysis of Arabic word roots based on four famous Arabic dictionaries. This Chapter also discusses the analysis of Arabic patterns and affixes. Finally it deals with the formation process of Arabic words.

CHAPTER 5 This chapter presents the **Phase I** of the developed solution. It presents the computational model solution for the extraction Arabic word roots. This chapter illustrates the overall architecture of the proposed computational model. Also, it presents and discusses all the supplementary modules that assist the process of the computational model. The pseudo code algorithm of **Phase I**, and the detailed discussions of the algorithm steps are also presented in this chapter.

CHAPTER 6 This chapter presents the **Phase II** of the developed solution which is based on the artificial neural networks. The chapter discusses the neural network methodology used in this study. It gives a detailed description of the system prototype, architecture and implementation. Furthermore, it presents the experiments that are used in order to optimise the neural network parameters. The corpus that is used in this study and its annotated also presented in this chapter. Finally, the chapter deals with the extraction of Arabic word roots as a classification problem and the artificial neu-

ral network as a classifier tool.

CHAPTER 7 This chapter deals with the testing and the evaluation of the proposed computational model. The two phases, testing experiments and their discussions, are presented in this chapter.

CHAPTER 8 This chapter is the last chapter of the thesis. It is a summary of the work which has been carried out in the current study. It also shows the main findings of the system evaluation and attempts to answer the research question and ends with some suggestions for future work to be done in the research area.

1.8 The Roadmap of the Developed Approach

As was pointed earlier, the main aim of this study is to develop a root-stemming algorithm for Arabic language giving only one root solution. This is done by investigating the viability of applying a computational and a neural networks approach, avoiding having a pattern and an affix lexicons and a set of linguistics rules.

The developed MUAIDI-STEMMER algorithm can be viewed as a single system has two main components. The first component which is called **Phase I** tries to extract the root of a given word by investigating the most likely Arabic root-letters and excluding the affixes (non-root) letters. If the first component is failed to extract the root, then the second component is triggered. The second component which is called **Phase II** is based on the neural networks.

Figure 1.1 depicts the main elements "roadmap" of the MUAIDI-STEMMER approach. This figure shows the chapters in which each element is described in detail. Also, the figure depicts the relationships among these elements.

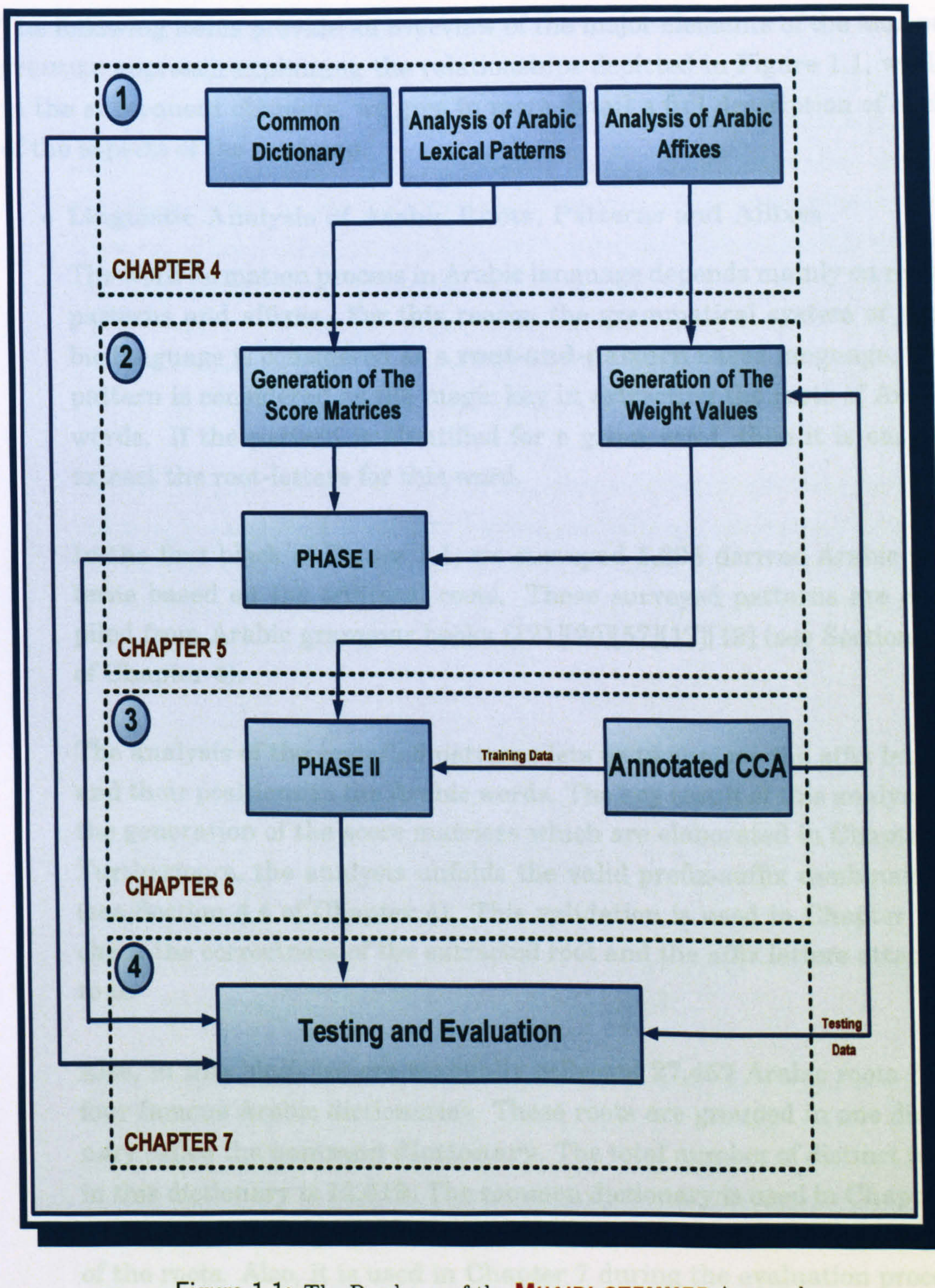


Figure 1.1: The Roadmap of the **MUAIDI-STEMMER** Approach

The following items provide an overview of the major elements of the MUAIDI-STEMMER approach explaining the relationships depicted in Figure 1.1, while in the subsequent chapters, we give in more detail a full description of each of the aspects of the roadmap.

- Linguistic Analysis of Arabic Roots, Patterns and Affixes

  The word formation process in Arabic language depends mainly on roots, patterns and affixes. For this reason the grammatical system of Arabic language is considered as a **root-and-pattern** based language. The pattern is considered as the magic key in extracting the roots of Arabic words. If the pattern is identified for a given word, then it is easy to extract the root-letters for this word.

  In the first block of Figure 1.1, we surveyed **1,893** derived Arabic patterns based on the triliteral roots. These surveyed patterns are compiled from Arabic grammar books [121][20][57][17][19] (see Section 4.3 of Chapter 4).

  The analysis of the compiled patterns lets us to analyse the affix letters and their positions in the Arabic words. The key result of this analysis is the generation of the score matrices which are elaborated in Chapter 5. Furthermore, the analysis unfolds the valid prefix-suffix combinations (see Section 4.4 of Chapter 4). This validation is used in Chapter 7 to check the correctness of the extracted root and the affix letters attached to it.

  Also, in this block we are manually collected **27,457** Arabic roots from four famous Arabic dictionaries. These roots are grouped in one dictionary called the **common dictionary**. The total number of distinct roots in this dictionary is **12,619**. The common dictionary is used in Chapter 6 during the building-up of the annotated corpus to check the validation of the roots. Also, it is used in Chapter 7 during the evaluation process for the same reason.

- The Computational Approach Subsystem (Phase I)

  The second block of Figure 1.1 is for **Phase I** which is based on the analysis of Arabic patterns and the affix letters. In this phase we assigned a weight value (Chapter 5) to every letter in the tested word then multiply these values by the score values (Chapter 5). The weight values are numeric values based on the frequency analysis of Arabic affix letters, while the score values are based on the analysis of the Arabic patterns, which show the possibility of the letter to be an affix letter. If there are three zeros in the resultant multiplications then the corresponding letters of these zeros represent the root-letters.

  If there are just two zeros in the resultant multiplications, then the root-distance is computed, which counts the gap between these zeros (number of letters). If there are two letters or more, then one of these letters is a root-letter, which is represent the minimum weight value. **Phase I** is elaborated in Chapter 5.

- The Subsystem of Neural Networks Approach (Phase II)

  The third block of Figure 1.1 is for **Phase II** which is based on artificial neural networks trained by the backpropagation learning rule. In this developed phase, we formulated the root extraction problem as a classification problem and the neural network as a classifier. As mentioned at the beginning of this chapter, **Phase II** is triggered when **Phase I** is failed to extract the root for a given Arabic word.

  The dataset used to train and to test the developed neural networks of **Phase II** are extracted from the annotated corpus of contemporary Arabic (Chapter 6). The training dataset is utilised for optimising the neural networks and for selecting their parameter values, whereas the test datasets is used solely in the final evaluation of the developed neural networks (Chapter 7). **Phase II** is elaborated in Chapter 6.

- The Annotated Corpus of Contemporary Arabic

  The right object of block three of Figure 1.1 represents the annotated Arabic corpus. The **Corpus of Contemporary Arabic** [29][30] is chosen to be the main source for data to conduct this research. Such a choice is based on factors like size, availability and diversity of content. The **Corpus of Contemporary Arabic** is compiled by Latifa Al-Sulaiti and Eric Atwell. This corpus needs to be processed (annotated) before it is carrying out; the processing includes assigning the morphological features to every valid Arabic word in the corpus. The output of this process is the generation of the **Annotated Corpus of Contemporary Arabic**. This generated annotated corpus is used for training the developed neural networks which are presented in Chapter 6 as well as it is used in Chapter 7 to check the performance and the quality of the developed algorithm. A full description of the corpus of contemporary Arabic and the compiled annotated corpus are presented in Section 6.3 of Chapter 6.

- Testing and Evaluation

  The fourth block of Figure 1.1 deals with the testing and the evaluation of the developed stemmer. The full descriptions of these testing and evaluations are elaborated in Chapter 7.

Chapter 2

ARABIC LANGUAGE AND MORPHOLOGY SYSTEM

2.1 Introduction

This chapter introduces the relevant basic elements of the Arabic language. This covers the Arabic alphabets, the structure of the Arabic language, and the features of the Arabic writing system. The derivational and inflectional morphology of Arabic language is presented in this chapter. The Arabic word classes, i.e. nouns, verbs, and particles are also discussed in the last section of this chapter. A non-Arabic reader will find this introduction is useful and interesting, but this is far from being a comprehensive Arabic guide. Further reading is recommended such as [124][81][77].

If the reader have had adequate background with the subjects mentioned above, then they may prefer to skim the chapter quickly.

2.2 Overview of Arabic Language

Arabic is an international language and one of the official languages in the United Nations. Arabic belongs to the Semitic languages family [136] (dif-

ferent from Indo-European languages in some respects). This family also includes Akkadian, Aramaic, Ethiopic, Hebrew, Phoenician, Syriac and Ugaritic. Arabic is considered the main representative of the South-Central Semitic language group.

Today, Arabic is the sixth most widely spoken language in the world [4]. The estimated number of Arabic speakers is over **250** million, of which roughly **200** million are first language speakers and **50** million are second language speakers[1]. Arabic is an official language in over **22** countries. It is spoken as first language in North Africa (Algeria, Egypt, Libya, Morocco, Tunisia, Sudan), the Arabian Peninsula (Bahrain, Kuwait, Oman, Qatar, Saudi Arabia, United Arab Emirates, Yemen), Middle East (Iraq, Jordan, Lebanon, Syria), and other Arab countries (Mauritania, Comoros, Djibouti, Somalia). Since Arabic is the language of the Quran, the holy book of Islam, it is also spoken as a second language by several Asian countries such as: Indonesia, Pakistan, Iran, Uzbekistan and Malay.

2.3 Arabic Alphabets

The Arabic script was derived from a type of Aramaic via the Nabatean cursive alphabet [10], with the earliest known document dating from 512 AD. The Aramaic language has fewer consonants than Arabic, so new letters were created around the 7th century by adding dots to existing letters. A summary of the features of Arabic writing appears below.

- Arabic alphabet consists a core of twenty-eight letters in addition to **hamza** [همزة][2] (glottal stop) and two variants of existing letters (**alif** [ألف] and **ta** [تاء]). Each letter can appears in up to four different shapes, depending on the position of the letter in the Arabic word. The position may be at the beginning (initial), in the middle (medial form), or at the

[1]As stated in http://www1.georgetown.edu/departments/arabic/about/info/ [Accessed on 12-02-2008]

[2]The Arabic examples in this thesis were produced using the ArabTeX package for LaTeX2.

end of a word (terminal form). The letter may be also written separately (alone/isolation form). Figure 2.1 represents an example of letter **Kaaf**[ك] in the four different positions, while Table 2.1 lists all the Arabic alphabets in their various shapes.

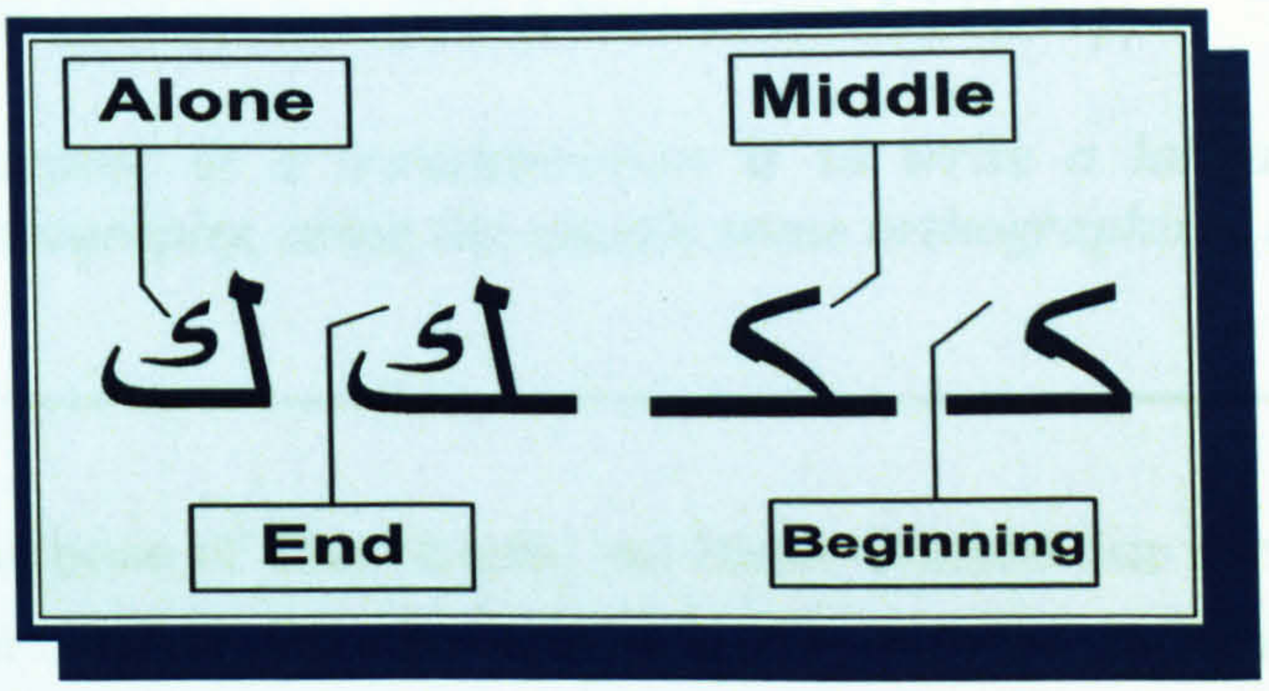


Figure 2.1: Example of Letter **Kaaf** [ك] in the Four Different Positions

Thus, a three letter word, for example, may start with a letter in beginning form, followed by a letter in medial form and, finally, by a letter in an end form like:

[كتب]

instead of:

[ك ت ب]

But the reality is even worse since a letter, in the middle of a word, may have the final or the initial form as in:

[عربي]

Because some letters do not connect with any character that comes after. They have only two forms: isolated (which is also used as initial) and final (also used as middle). Letters 8, 9, 10, 11 in Table 2.1 are examples of this type of letters.

- Many efforts [51][78][104] have been made at representing Arabic alphabets using Western character sets; this is usually referred to as transliteration. Beesley [45] defines the purpose of transliteration as follows:

> **DEFINITION 2.1** (TRANSLITERATION)
>
> *The purpose of a transliteration is to write a language in its customary orthography, using the exactly same orthographical conventions. [45]*

For the purpose of this thesis, we have defined our own transliteration scheme for Arabic alphabets, which is presented in Table 2.1. Each Arabic letter in this scheme is mapped to only one English letter.

Wherever in this thesis, any Arabic word is annotated as a triple attributes to be more clear for a non-Arabic reader. The first attribute for the Arabic word itself which is written in Arabic scripts between two square brackets, the second attribute for an English transliteration which is written in italics, while the third one for English translation which is written between two quotation marks. Figure 2.2 shows an example.

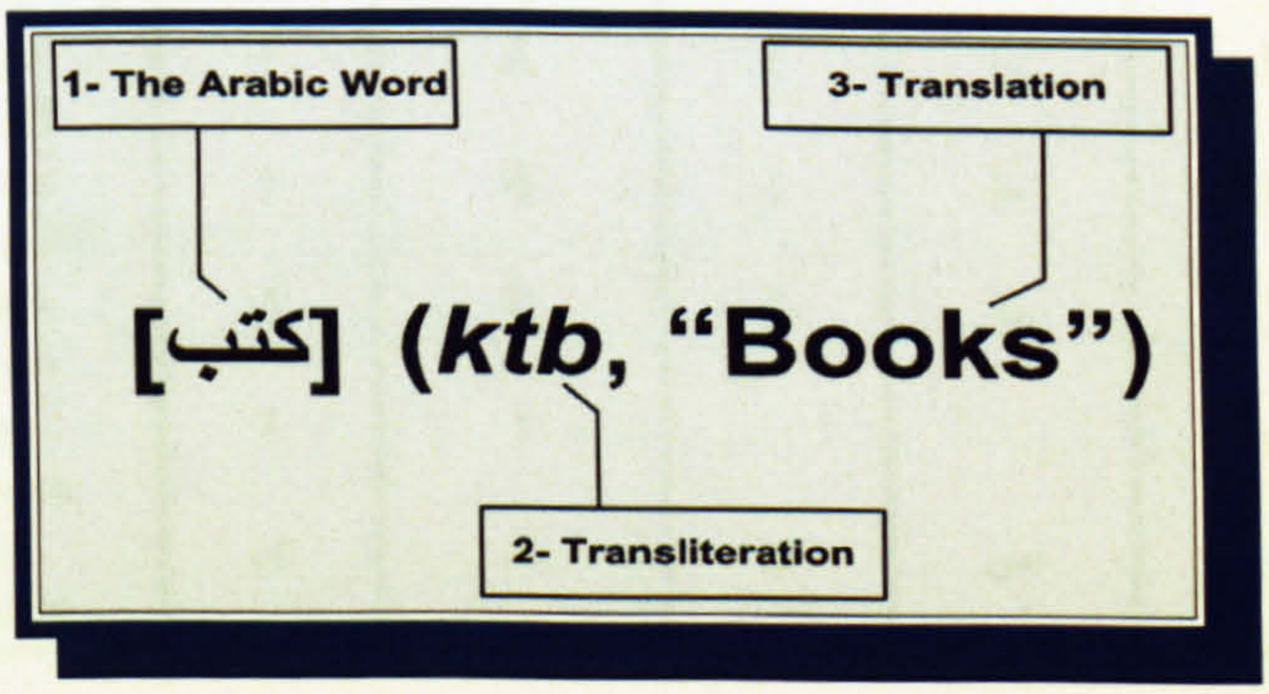


Figure 2.2: An Example of Annotated Arabic Word

No.	Name	Alone	Translit-eration	Initial Form	Meidal Form	Terminal Form	Sound
1	Alef	ا	A	ا	ـا	ـا	A
2	Baa	ب	b	بـ	ـبـ	ـب	B
3	Taa	ت	t	تـ	ـتـ	ـت	T
4	Thaa	ث	t̂	ثـ	ـثـ	ـث	Th
5	Jiim	ج	j	جـ	ـجـ	ـج	J
6	Haa	ح	H	حـ	ـحـ	ـح	Ha
7	Khaa	خ	Ĥ	خـ	ـخـ	ـخ	Kh
8	Daal	د	d	د	ـد	ـد	D
9	Thaal	ذ	ḋ	ذ	ـذ	ـذ	Th
10	Raa	ر	r	ر	ـر	ـر	R
11	Zaay	ز	z	ز	ـز	ـز	Z
12	Siin	س	s	سـ	ـسـ	ـس	S
13	Shiin	ش	ŝ	شـ	ـشـ	ـش	Sh
14	Saad	ص	S	صـ	ـصـ	ـص	S
15	Daad	ض	Ṡ	ضـ	ـضـ	ـض	Tha
16	Taa	ط	T	طـ	ـطـ	ـط	Ta
17	Thaa	ظ	t̂	ظـ	ـظـ	ـظ	Tha
18	Ayn	ع	’	عـ	ـعـ	ـع	a’a
19	Ghayn	غ	g	غـ	ـغـ	ـغ	Gh
20	Faa	ف	f	فـ	ـفـ	ـف	F
21	Qaaf	ق	q	قـ	ـقـ	ـق	Q
22	Kaaf	ك	k	كـ	ـكـ	ـك	K
23	Laam	ل	l	لـ	ـلـ	ـل	L
24	Miim	م	m	مـ	ـمـ	ـم	M
25	Noon	ن	n	نـ	ـنـ	ـن	N
26	Haa	ه	h	هـ	ـهـ	ـه	H
27	Waaw	و	w	و	ـو	ـو	W
28	Yaa	ي	y	ي	ـي	ـي	Y
29	Hamza	ء	’	ء	ء	ء	’
30	Hamza on Alef Maqsura	ئ	Ã	ئـ	ـئـ	ـئ	’
31	Hamza on Alef	أ	Ā	أ	ـأ	ـأ	A’
32	Hamza below Alef	إ	A̲	إ	ـإ	ـإ	E’
33	Hamza on waw	ؤ	w̄	ؤ	ـؤ	ـؤ	W’
34	Taa Marbuta	ة	t			ة,ـة	t
35	Alef Maqsura	ى	A	ى	ـى	ـى	A

Table 2.1: Arabic Alphabets

- Fifteen of the Arabic letters contain dots. These dots may be one, two or three, either above, below or in the middle of a letter. Ten Arabic letters have one dot, three letters have two dots, while two letters have three dots [42]. The dots are used to differentiate between a set of consonants that have the same shape. Letters 2, 3, and 4 in Table 2.1 are examples of dot letters in Arabic language.

- Three letters from the twenty-eight letters appear in a different shapes, which are they:

  1. Hamza [ء]: This shape can be: on Alef [أ] (Ā), below Alef [إ] (Ą), on Waaw [ؤ](w̄), on Alef Maqsura [ئ](Ã), or isolated [ء](’). Letters 29 – 33 in Table 2.1.
  2. Taa-Marbuta [ة]: This is a special form of the letter [ت](t), it always appears at the end of the word. Letter 34 in Table 2.1.
  3. Alef-Maqsura [ى]: This is a special form of the letter [ا](A), it always appears at the end of the word. Letter 35 in Table 2.1.

  The above three letters pose some difficulties when building morphological systems. Many of the written Arabic texts and Arabic web sites ignore the **Hamza** and the two dots above the **Taa-Marbuta**. For example, the Arabic word [مدرسة] (mdrst,“school”) may appear in many texts as [مدرسه] (mdrsh) (which means “school” or “his teacher”) without two dots above the last letter. When comparing the last letter in the two previous words, we found it was [ة](T) in the first word, while it was [ه](H) in the second word.

- Twenty-five of Arabic alphabets represent consonants. The remaining three letters represent the weak letters or the long vowels of Arabic (shortly vowels). These letters are: **Alef**[ا], **Waaw**[و] and **Yaa**[ي].

- Three short vowels can be added to the letters set in Arabic language. These letters are called diacritics (vowelisation). Diacritics are not considered as Arabic Letters. They are just marks normally placed above or under Arabic letter. Diacritics make the reading of Arabic words less ambiguous. For example, the word [كتب] which is transliterated as **ktb**,

can be read in two different ways. If **ktb** is read as [كَتَبَ] (**kataba**), in this case **ktb** is a verb and means "he wrote", whereas, if **ktb** is read as [كُتُبْ] (**kutub**), it means "books".

Today, diacritics are used in very special documents such as in the Quran, the holy book of Islam, to prevent misunderstandings and misreadings. Also, diacritics are used in political texts, children's schoolbooks and poetry books. However, most recent written Arabic texts are non-vowelised.

The three short vowels (diacritics) in Arabic are summarised as follows:

1. Fatha [فتحة] above a consonant [ـَـ], is pronounced like the letter **a** in cat. For example, [كَتَبَ] (**kataba**, "he wrote").
2. Damma [ضمة] above a consonant [ـُـ], is pronounced like the letter **u** in put. For example, [كُتُب] (**kutub**, "books").
3. Kasra [كسرة] below a consonant [ـِـ], is pronounced like the letter **i** in bit. For example, [طالِب] (**Talib**, "student").

- Arabic has some special marks rather than the previous diacritics. One of these marks is called **gemination** mark (**shaddah** [شدة] or **tashdeed**). Gemination is a mark written above the letter [ـّـ] to indicate a doubled consonant while pronouncing it. This is done when the first consonant has the null diacritical mark **skoon** [ـْـ], and the second consonant has any other diacritical mark. For example in the Arabic word [كسْسَر] (kssr,"he smashed to pieces"), when the first syllable ends with [س](s) and the next starts with [س](s), the two consonants are united and the gemination mark indicates this union. So, the previous word is written as [كسّر], and it has four letters {ك، س، س، ر}.

Unfortunately, Arabic people do not explicitly mention the gemination mark in their writing. They depend on their knowledge of the language to supply the missing letter and write the words without it. In consequence, this is make the morphology process of such words is not an easy task.

- As is the case with all Semitic languages, Arabic alphabets are written and read from **right-to-left** with a cursive consonantal script. Letters are mostly connected and there is no capitalisation in Arabic language.

- Many languages are written with, or used to be written with the Arabic script [16]. This includes: Afrikaans, Albanian, Arabic, Azeri, Baluchi, Berber, Bashkir, Belarussian, Bosnian, Chaghatai, Chechen, Comorian, Fulani, Hausa, Kashmiri, Kazakh, Kurdish, Kyrghyz, Malay, Mandinka, Morisco, Mozarabic, Nubian, Pashto, Persian/Farsi, Punjabi, Sanskrit, Sindhi, Somali, Songhay, Swahili, Tamazight, Tatar, Turkish, Turkmen, Urdu, Uyghur, Uzbek and Wolof.

2.4 Morphology

The main source of word variation in a language comes form its morphology [34]. Morphology[3] is the field of linguistics that studies the internal structure and formation processes of words [88]. The internal structure of a given word is its **morphemes** [82][88]. **A morpheme** is often defined as the smallest meaningful and significant unit of language, which cannot be broken down into smaller parts [88]. So for example, the word **book** consists of a single morpheme (the morpheme **book**), while the word **books** consists of two morphemes: the morpheme **book** and the morpheme **-s** (indication of plural). In Arabic language for example, the word [علمهم]('lmhm,"he taught them") consists also of two morphemes the verb [علم]('lm,"he taught") and the pronoun [هم](hm,"them").

According to the previous examples, there are two types of morphemes: **roots** and **affixes**. The root is the main morpheme of the word, supplying the main meaning, while the affixes add additional meaning of various kinds. More details about Arabic roots and affixes are explained in Chapter 4.

In more general morphemes could be classified as: (1) free morphemes and (2) bound morphemes, Figure 2.3 illustrated this classification. Free morphemes

[3]Is derived from the Greek word *morphê*, (which means form or shape).

are units of meaning which can stand on their own as words (independent words). Bound morphemes are also units of meaning; however, they can not occur as words on their own; they need to be attached to something such as free morphemes. Bound morphemes are also referred as affixes [117]. There are three types of affixes in Arabic language: prefixes, infixes, and suffixes. Section 4.4 of Chapter 4 presents more highlights of these affixes.

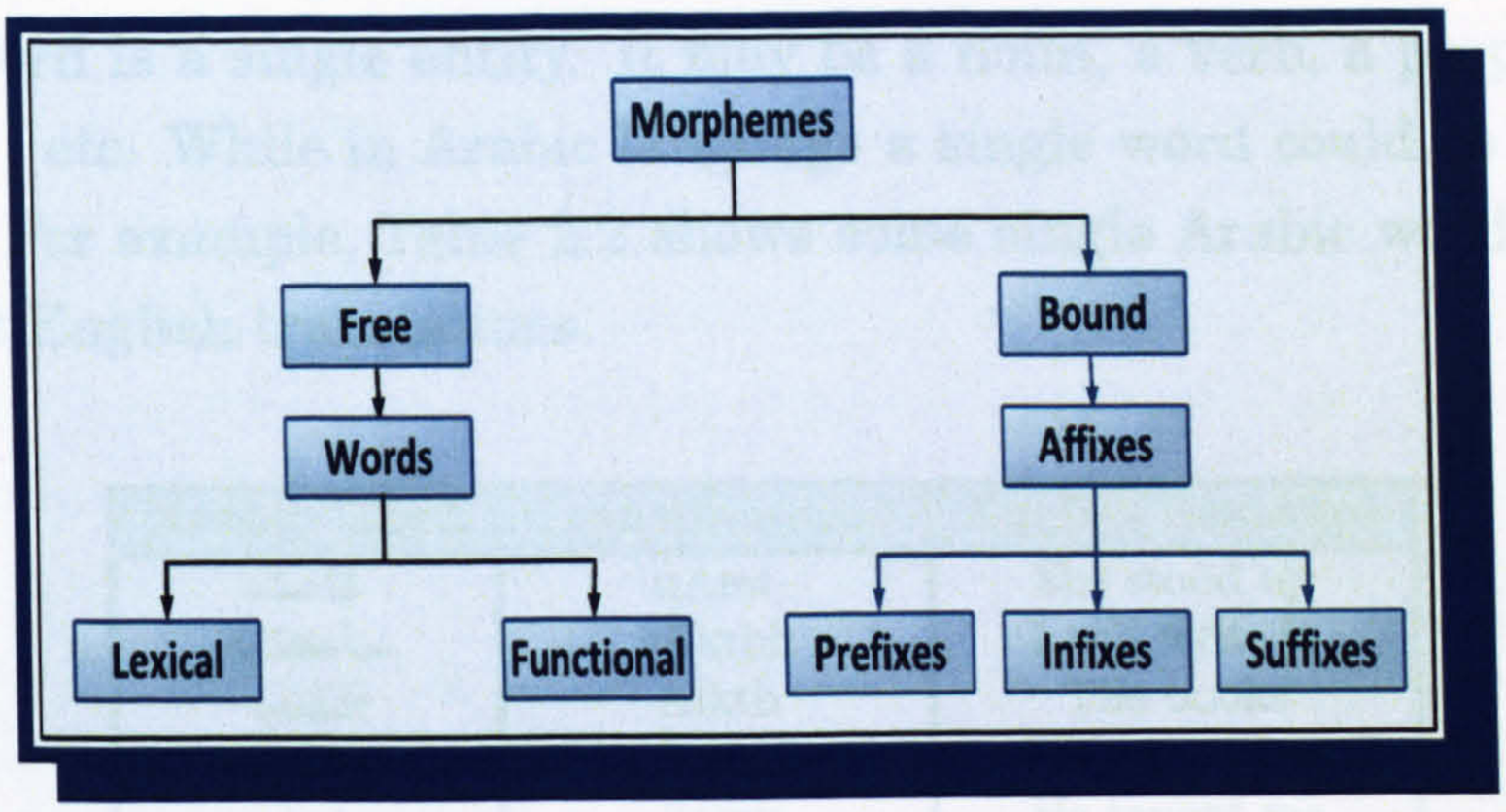


Figure 2.3: Morpheme Classes

The free morphemes (which also refer to as unbound morphemes) may be sub-grouped into two categories. The **first category** is called lexical morphemes, which covers the words in the language carrying the content of the message. Examples from English language: **book**, **compute**, and **write**, while examples from Arabic language: [**قرأ**] (qrĀ,"read"), [**لعب**] (l'b,"play"), and [**كتب**] (ktb,"write").

The **second category** is called functional morphemes, which covers the function words in the language. In literature, these words also refer to **stopwords**. The function words include adverbs, prepositions, pronouns, conjunctions, and prepositions. Examples from English language: **on**, **that**, **the**, and **above**. Examples from Arabic language: [**في**] (fy,"in"), [**فوق**] (fwq,"above"), and [**تحت**] (tHt,"under"). Section 5.2.3 of Chapter 5 presents more details for these words, while Appendix C lists a set of stopwords compiled through this

study.

2.5 Arabic Morphology

Arabic morphology represents a special type of morphological system. Arabic has a rich and complex morphology [101]. In many cases, one orthographic word consists of many semantic and syntactic words [92]. In English language a word is a single entity. It may be a noun, a verb, a preposition, an article, ..., etc. While in Arabic language a single word could be a complete sentence. For example, Table 2.2 shows some single Arabic words and their equivalent English translations.

Arabic Word	Transliteration	English Sentence
قامت	qAmt	She stood up
سأكتبه	sĀktbh	I will write it
الكتب	Alktb	The books
رأيناه	rĀynAh	We saw him
علمني	'lmny	He taught me
فانصرف	fAnSrf	Then he departed
سأعطيك	sĀ'Tyk	I will give you

Table 2.2: Example: An Arabic Word could be a Complete English Sentence

Traditionally there are two types of morphology in Arabic Language, derivational morphology and inflectional morphology [63]. There is no definite agreement amid Arab linguists about the exact meaning and using of these terms in Arabic language. Some linguists used the term morphology as a synonym of derivation, while some others use the term inflection as a synonym of derivation. However, most of them are agreed that morphology is concern in the studying of word formation. The next two sections deal in detail with these terms.

2.5.1 Derivational Morphology

Crystal [56] defines the derivation as:

DEFINITION 2.2 (DERIVATION)

Derivation is a term used in morphology to refer to one of the two main categories of process of word formation (Derivational Morphology), the other being (Inflectional). ... Basically, the result of derivational process is a new word (e.g. nation → national), whereas the result of inflectional process is a different form of the same word (e.g. nations → nationals). ([56] : page 111).

Thus, derivational morphology deals with the formation of new words from existing roots using derivational affixes. Derivational morphology often changes the grammatical word class (part-of-speech) and/or the basic meaning of the word. To simplify the definition of the derivational morphology, consider the English words: **computed**, **computer**, **computerise**, **recompute**, **computation**. These words are related by derivation since they are different words based on the same root **compute**. In the first word, the suffix (**ed**) is added, in the second one, the suffix (**er**) is added and changed the class of the word to noun. In the third word, the prefix (**re**) is added, while in the fourth word, the suffix **ation** is added.

In Arabic language, the derivational morphology is the process of interdigitation of roots with the morphological patterns. The morphological pattern is a template or a paradigm showing the positions of the root-letters where they should be. More details about morphological patterns are presented in Chapter 4. Figure 2.4 shows seven derived words from the Arabic root [كتب](ktb,"write").

2.5.2 Inflectional Morphology

Inflectional morphology involves adding inflectional affixes in order to produce different forms of the same word to give it extra meaning. This extra meaning deals with the grammatical features of the languages such as per-

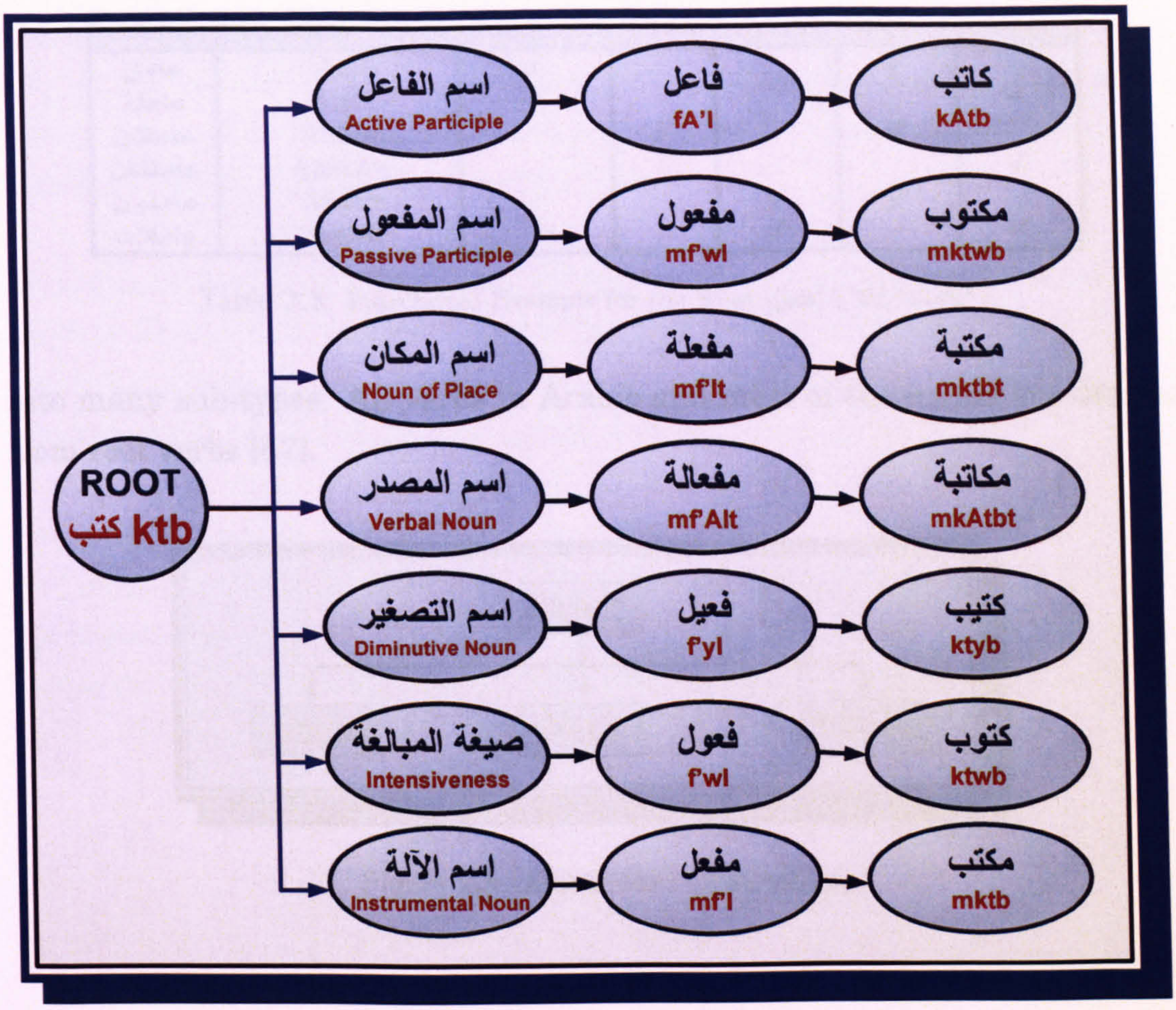


Figure 2.4: A Derivational Morphology Example for the Word [كتب](*ktb*, "write")

son, number, gender, tense, case, and mode. Inflectional morphology does not change the class or the part-of-speech of the word. Also, it does not change the core meaning of the word. Table 2.3 shows six different Arabic word forms that are inflected from the same root [عمل] ('ml,"work").

2.6 Classification of Arabic Words

Arabic words can be classified into three main categories according to the part-of-speech [17]: noun [اسم] (**Asm**), verb [فعل] (**f'l**), and particle [حرف] (**Hrf**). Figure 2.5 illustrates these categories. Each main category can be subdivided

Noun	Transliteration	Singular	Dual	Plural	Masc.	Femn.
عامل	'Aml	✓			✓	
عاملة	'Amlt	✓				✓
عاملان	'AmlAn		✓		✓	
عاملتان	AmltAn		✓			✓
عاملون	'Amlwn			✓	✓	
عاملات	'AmlAt			✓		✓

Table 2.3: Inflectional Example for the Root [عمل] ('*ml*,"work")

into many sub-types. All verbs in Arabic and most of the nouns are derived from root verbs [57].

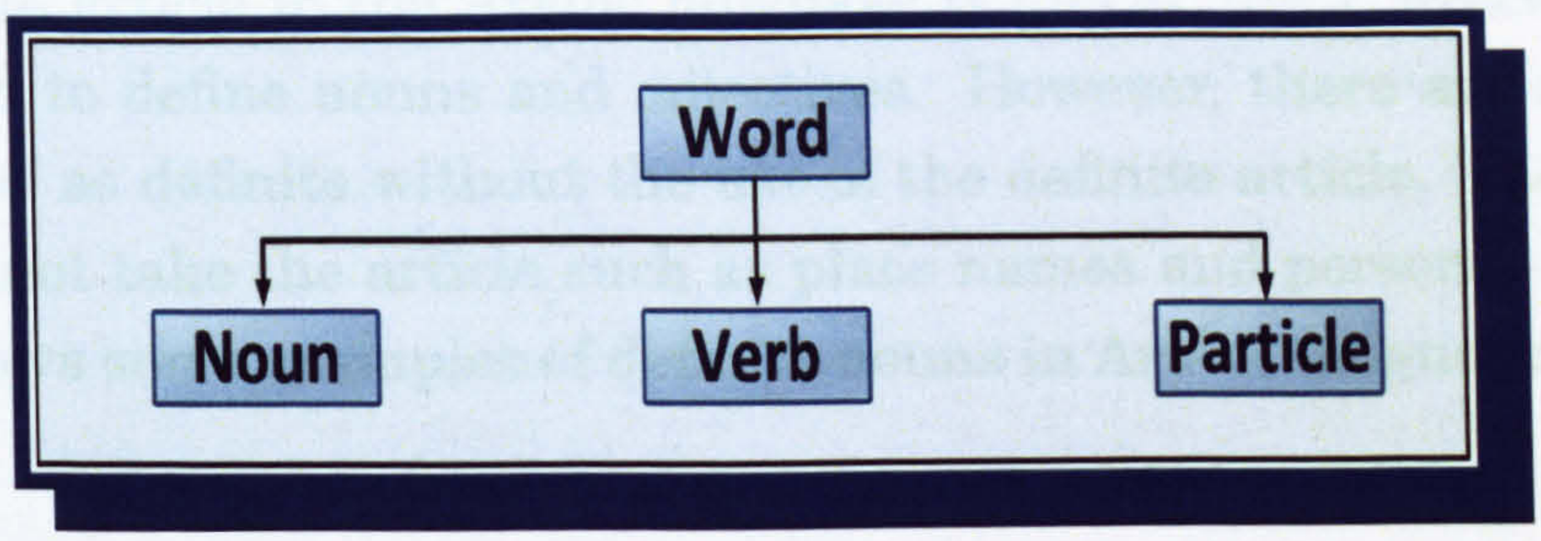


Figure 2.5: Arabic Word Classification

2.6.1 Arabic Nouns

Arabic nouns are words used to name or denote a class of things, places, people, ideas, or qualities. This includes adjectives, pronouns, number (singular, dual, and plural), case (nominative, genitive, and accusative), definiteness/indefiniteness, and even prepositions. The Arabic noun may be attached with three prefixes and three suffixes. In various languages, they are marked, by affixes or particles, as to their number, gender, definiteness, and especially cases.

Nouns may be classified as simple [بسيط] or derived [مشتق]. A simple noun is one that is not derived from another word; it does not refer to a verbal root (e.g., [أسد]($\bar{A}sd$,"lion")). On the other hand, derived nouns are taken from another word, usually a verb they have a root to refer to. A derived noun is

usually close to its root in meaning (e.g., [مدرسة](*mdrst*,"school")). It indicates, besides the meaning, the concrete thing that caused its formation (case of the agent-noun), or underwent its action (case of the patient-noun), or any other notions of time, place, or instrument [7][8].

Nouns may be classified as inflected [معرب], which have variable diacritical endings due to their functional position in a sentence, or uninflected [مبني], which have constant diacritical endings regardless of their position in a sentence. Nouns may have masculine [مذكر] or feminine [مؤنث] gender.

The definite article in the Arabic language is [ال](*Al*,"the"), which is written as prefixed to define nouns and adjectives. However, there are names that are counted as definite without the use of the definite article. These types of nouns cannot take the article such as place names and personal names. Table 2.4 shows some examples of definite nouns in Arabic language.

Arabic Word	Translation
[الصيف](*AlSyf*)	the summer season
[لبنان](*lbnAn*)	country name
[حسن](*Hsn*)	personal name
[الطالب](*AlTAlb*)	the student

Table 2.4: Examples of Definite Nouns

Nouns may have singular, dual, or plural number. Dual and plural nouns can be formed by adding suffixes to a singular noun, but some plurals are irregular in their grammatical formation.

2.6.2 Arabic Verbs

Verbs are words which usually express an action taken by something, the state of being, or an interaction between one thing and another. Like nouns,

there are many variations of verbs.

Most verbs in the Arabic language follow clear rules that define their morphology and generate their paradigms. The Arabic verb has three numbers: singular, dual, and plural. Arabic verb also has two genders: masculine and feminine. As in other languages, there are three persons for the verb in Arabic language. The first person has no dual, and both its singular and plural are of the common gender. Arabic language, in common with other Semitic languages, is deficient in tenses. Arabic language has basically three tenses: perfect, imperfect, and imperative. Perfect tense [ماضي](*mAṠy*,"past") denotes actions completed and may be translated into English by the past or the perfect, while imperfect [مضارع](*mṠAr'*,"present") denotes uncompleted actions. The imperfect is most frequently translated into English by the present. For instance, [كتبَ](*ktba*,"he wrote") indicates the perfect tense, while [تكتبُ](*tktbu*,"she is writing") indicates the imperfect tense of the feminine.

In the perfect Arabic tense, the different persons are expressed by suffixes, while the imperfect bas both prefixes and partly suffixes to denote number and gender. Table 2.5 shows an example of the full form of the imperfect indicative of the Arabic verb [كتب](*ktb*,"he wrote").

Person	Gender	Singular	Dual	Plural
1^{st}	masculine	[أكتب](*Aktb*,"I write")	-	[نكتب](*nktb*)
1^{st}	feminine	[أكتب](*Āktb*,"I write")	-	[نكتب](*nktb*)
2^{nd}	masculine	[تكتب](*tktb*,"You write")	[تكتبان](*tktbAn*)	[تكتبون](*tktbwn*)
2^{nd}	feminine	[تكتبِ](*tktbi*,"You write")	[تكتبان](*tktbAn*)	[تكتبن](*tktbn*)
3^{rd}	masculine	[يكتب](*yktb*,"He write")	[يكتبان](*yktbAn*)	[يكتبون](*yktbwn*)
3^{rd}	feminine	[تكتب](*tktb*,"She write")	[تكتبان](*tktbAn*)	[يكتبن](*yktbn*)

Table 2.5: Full Form of the Imperfect Indicative of the Arabic Verb [كتب](*ktb*,"he wrote")

2.6.3 Arabic Particles

The particle in Arabic is called [حرف](**hrf**), meaning a letter. This category includes the remaining words, and their function is to assist other words in their semantic function in the sentence [20]. It is defined according to its functional category such as; preposition, conjunction, adverbs, interrogative particles, exceptions, and interjections [92]. Some Arabic particles are joined to other words, such as [ب](**b**, "in") as [بالمدينة](bAlmdynt,"in the city") and sometimes it means (by [بال]) as [بالعمل](bAl'ml,"by the work") or means (for [ل]) as [للجامعات](lljAm'At,"for universities"). This joining particles creates problems for information retrieval in Arabic. Therefore, any efficient Arabic stemmer should deal with this type of particles.

The other type of particles is a standalone particle, namely [في]*fy*,"in") and [من](*mn*,"from"). These particles are considered as stopwords in Arabic language, and there is no need to discuss them in detail here. The definition of stopwords and the process of extracted such words is discussed at length in Chapter 5, while Appendix C lists all the extracted stopwords for Arabic language.

2.7 Summary

Arabic language is an international language belonging to the Semitic languages family (different from Indo-European languages in some respects). The Arabic alphabet consists of twenty-eight letters in addition to some variants of existing letters. Each letter can appear in up to four different shapes, depending on the position of the letter in the Arabic word. Twenty-five of Arabic letters represent consonants. The remaining three letters represent the long vowels of Arabic. The Arabic writing system goes from right to left and most letters in Arabic words are joined together.

Arabic has a rich and complex morphology. In many cases, one orthographic word is comprises many semantic and syntactic words. Traditionally there are two types of morphology in Arabic Language: derivational morphology

and inflectional morphology. The derivational morphology deals with the formation of new words from an existing words while, the inflectional morphology deals with producing different forms of the same word to give it extra meaning.

All Arabic words could be classified into three main categories according to the part-of-speech : noun, verb, and particle. The noun and verb in Arabic might be further divided according to: number (singular, dual and plural), and case (nominative, genitive and accusative).

This chapter is far from being a comprehensive guide to Arabic language, it was covered just those elements which they thought are related to this research. The next chapter discusses the literature review of the study.

The next chapter presents the literature review of the study.

Chapter 3

LITERATURE REVIEW

3.1 Introduction

Stemming is a well studied problem in the field of natural language processing for both linguistic and computational purposes [139]. This chapter describes this problem in more detail, and presents the different strategies for dealing with it.

The contents of this chapter fall into two parts. The **first part** (Section 3.2 to Section 3.5) deals with the different techniques used in developing stemming algorithms. This part starts by defining the stemming concept. It continues by discussing the uses of stemming in natural language processing applications. Then, it proceeds with the classifications of stemming approaches for English and Arabic languages. Some of the previously developed stemmers for English and Arabic languages are also presented in this part.

The **second part** of this chapter deals with artificial neural networks (ANNs). ANNs have not been used previously in solving the stemming problem for Arabic languge. One of the main phases of the developed stemmer presented in this study is based on using ANNs. Thus, Sections 3.7 to 3.9 present a brief introduction to ANNs. This introduction covers the definition of ANNs, their topologies, the role and the types of activation functions, different learn-

ing paradigms and finally an overview of the backpropagation neural network learning rule. This part serves as a foundation for understanding the materials discussed in later Chapter 6, when ANNs using as a tool for solving the Arabic word root extraction problem.

3.2 Stemming Definition

Within languages such as English, there are many affixes added to words in order to change their tense and usage but often not their meaning. For example, the words **search**, **searching**, **research**, and **researcher** are four different forms derived from a base form **search**. The technique that mapped all these words to their base form is called **stemming**. The algorithm that does stemming is called a **stemmer**. Lovins [100] in 1968 defined stemming algorithm as:

DEFINITION 3.1 (STEMMING ALGORITHM)

A stemming algorithm is "a procedure to reduce all words with the same stem to a common form, usually by stripping each word of its derivational and inflectional suffixes" [100].

As the above definition suggests, a stemming algorithm aims to reduce the inflectional and derivational variants of words to a common form or base form. This base form is called **a stem** or **a root**. For example, Figure 3.1 presents **ten** English words were derived from the same stem (**comput**), and all of them share the same meaning.

Stemming is a complex process and it is a language dependent due to the fact that every language has its own morphological features and its large number of rules. A wide range of languages including Malay [137], Latin [70], Indonesian [41], Swedish [53], Dutch [134], German [64], Slovene [118], Bulgarian [109] and Turkish [62] have their own stemmers.

computer computable computation computal computally
computed computing computations computech computerise

Figure 3.1: An Example of English Words Derived from the Same Stem

The two terms **stem** and **root** could be used in English language interchangeably to point to the same thing. But, in Arabic language the matter is different. A **root** is the bare form of the verb giving the basic lexical meaning of the word. It is not derived from any other word. On the other hand, a **stem** is a combination of a root and derivational affixes. The stem is derived when the root is interdigited with the morphological patterns. Figure 3.2 shows these differences, and Chapter 4 will present these concepts in detail.

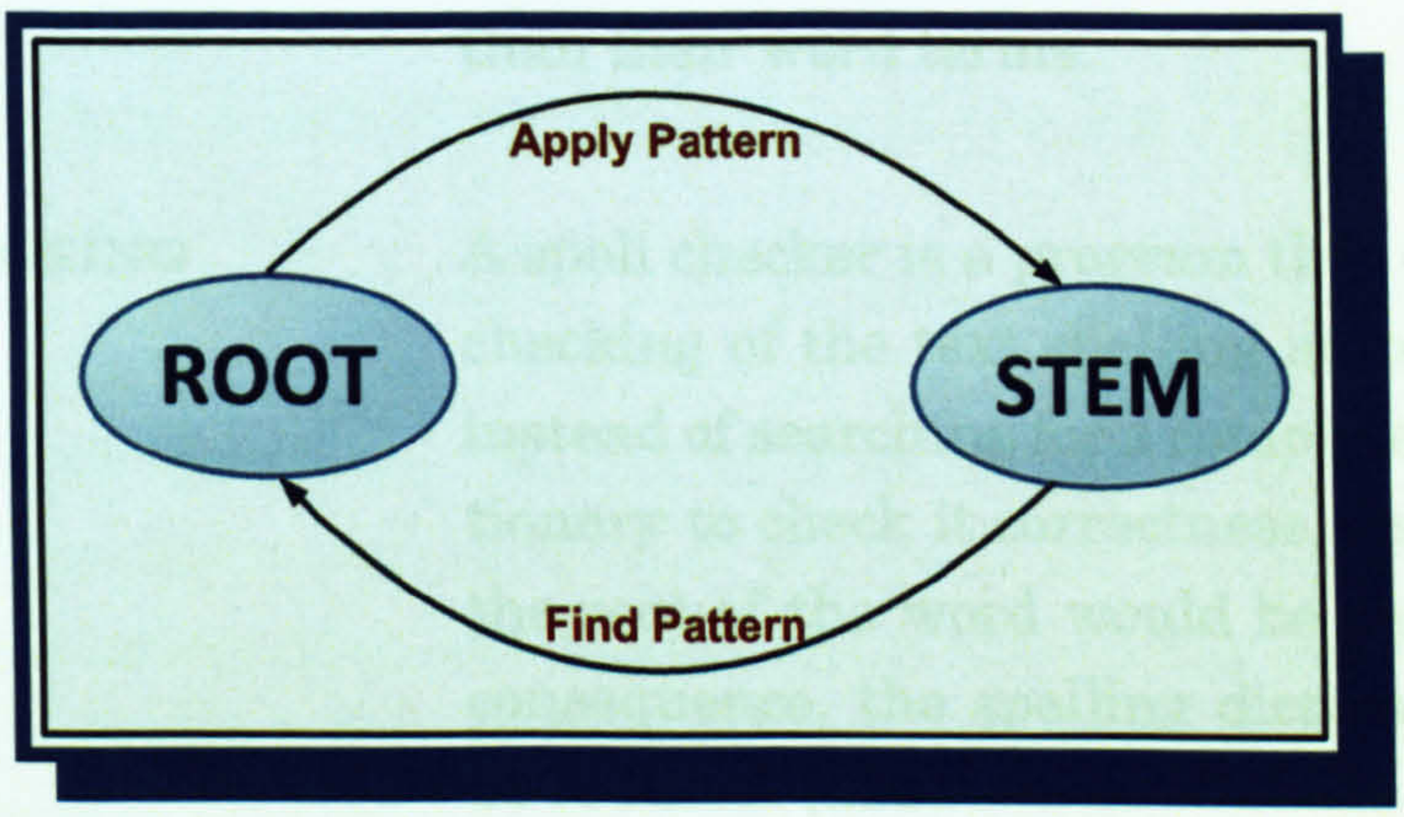


Figure 3.2: The **Root** Against the **Stem**

3.2.1 The Uses of Stemming

Stemming could be used in many natural language processing applications. The following list presents some of these applications.

INFORMATION RETRIEVAL Stemming performs two useful functions in information retrieval systems [99][96]. Firstly, it

may increase the number of relevant documents found. Since the stem of a term represents a broader concept than the original term. So, instead of using the original document or query terms, the stems of these terms are carried out for information retrieval. Secondly, the dictionary size that is used to represent the terms in these documents will be decreased. In consequence, this may result in a saving of storage space and processing time.

TEXT COMPRESSION

Text compression is a technique used to save disk storage and transmission time. Stemming is used here to reduce the texts size [74]. This is done by storing words in their root terms rather than their word terms.

SPELL CHECKING

A spell checker is a program that deals with the checking of the text spelling automatically. So, instead of searching for a complete word in a dictionary to check it correctness, only the stem or the root of the word would be searched for. In consequence, the spelling dictionary size could be reduced. [126].

DICTIONARY LOOKUP

Unlike English, which is a non-root based language, all Arabic words in an Arabic dictionary are sorted alphabetically depends on the roots of the words [73]. So, for example, when looking up to the Arabic word [معلم] (m'lm, "teacher") in an Arabic dictionary, it will be found under the group words started by the letter [ع](') , which is the first letter of the root [علم]('lm,"teach").

SPAM FILTERING

In this application, the emails are organised according to a specific criteria. The best example of using the stemming in this application is to reject all emails containing mentions of **sex**, so stemming could be here useful to detect any email contains any term derived from the word **sex**, in consequence the spam filtering system will reject such email [13].

PART-OF-SPEECH

In this application, the affixes that were resultant from the stemming process, could be used in determining the grammatical category or part-of-speech of the words, i.e. noun, verb, preposition, adjective, etc. [92].

3.3 Classes of Stemming Techniques for the English Language

In literature, there is no standard classification for stemming techniques for English language. Larkey et al. [96] divided English stemmers into two groups: strong and weak stemmers. Strong stemmers deal with removing a wide range of suffixes which tend to produce a large amount of conflation [96][28], while, weak stemmers deal with removing only inflectional suffixes [28]. Actually, these two groups are suffix removal stemmers with respect to the length of the suffixes that are removed.

Frakes et al. [67] presented another classification of English stemmers based on the strategies used in them. These strategies are: table lookup, n-gram, successor variety, and affix removal algorithms. Figure 3.3 shows Frakes et al. English stemmer classifications. The next subsections deal with them.

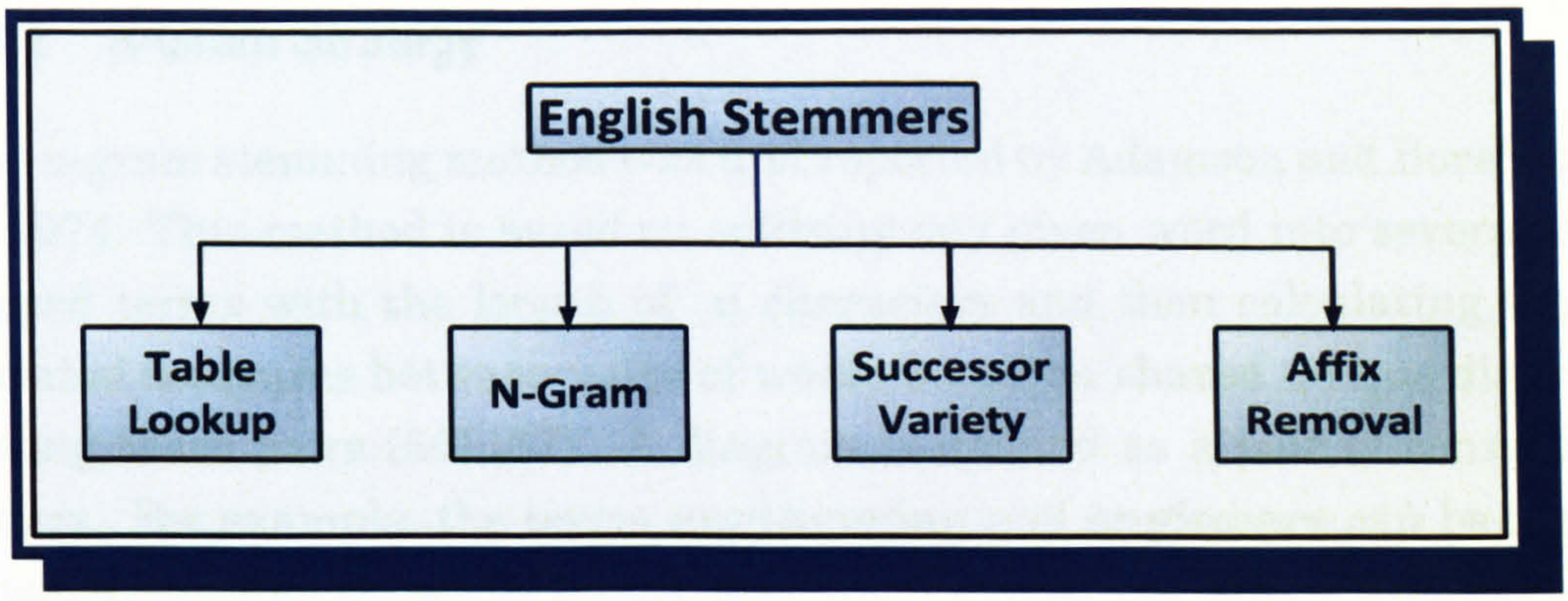


Figure 3.3: Classification of English Stemmers

3.3.1 Table Lookup Strategy

The table lookup stemming approach, also referred as dictionary-based [119], is the simplest of all known stemming approaches and is an easy one to implement. In this type of stemming, all the stems (with their possible syntactic varieties) are stored in a huge table or list. For a given word, it accesses the list and retrieves the associated root (stem). Consequently, the stems obtained are guaranteed to be highly accurate. An example of such a table is shown in Table 3.1.

Term	Stem
computer	comput
computational	comput
computation	comput
computable	comput
computationally	comput
computing	comput
.	.
.	.

Table 3.1: Table Lookup Example

However, the availability of such a table that should include all the English words is practically impossible [67]. Another disadvantage of this approach is the huge storage required to save such a table.

3.3.2 N-Gram Strategy

The n-gram stemming method was first reported by Adamson and Boreham [9] in 1974. This method is based on splitting any given word into several overlapped terms with the length of **n** characters and then calculating the associated measures between pairs of words based on shared unique diagrams among these pairs [66] [67]. A diagram is defined as a pair of consecutive letters. For example, the terms **engineering** and **engineers** can be broken into diagrams as shown in Fiqure 3.4.

engineering	→	*en ng gi in ne ee er ri in ng*
unique diagrams	→	*en ng gi in ne ee er ri*
engineers	→	*en ng gi in ne ee er rs*
unique diagrams	→	*en ng gi in ne ee er rs*

Figure 3.4: An Example of N-Gram Stemming

The example above, shows that the word **engineering** has **ten** diagrams, **eight** of them are unique. While, the word **engineers** has **eight** diagrams and all of them are unique. Both of these two words share the following **seven** unique diagrams: *en ng gi in ne ee er*. After the previous calculations are done for the word pairs, a similarity measure based on these calculations is computed. The dice's coefficient [132] is used as the **similarity measure (S)** and is defined as in Equation 3.1.

$$S = \frac{2C}{A+B} \tag{3.1}$$

Where **A** is the number of unique diagrams in the first word, **B** is the number of unique diagrams in the second word and **C** is the number of unique diagrams shared by the two words.

So, for the above example, the **similarity measure (S)** is calculated as shown

in Equation 3.2:

$$S = \frac{2 * 7}{8 + 8} = 0.875 \tag{3.2}$$

The resultant high similarity value **(0.875)** in Equation 3.2 indicates that the two words **engineering** and **engineers** are derived from the same stem. Since no stem is produced, we cannot call this as a **stemming strategy** [28]. It is a clustering procedure more than a stemming one. Furthermore, defining the similarity between words in a given text can be very time consuming. Despite this, the n-gram approach has some significant advantages such as:

- It is a statistically simple technique.
- No morphological rules or lookup tables are required.
- It is an independent language technique.

3.3.3 Successor Variety Strategy

The successor variety strategy was proposed by Hafer and Weiss [71] in 1974. This stemming technique tries to segment words into stems and affixes based on the determination of morpheme boundaries [24][71]. This is based on the statistical properties of successor letter counts. The motivation for using these quantities is the dependency of letters within words in a corpus, and for this reason the successor variety is considered as a corpus-based approach [109].

The successor variety of a string (prefix), with respect to the corpus used, is the number of distinct characters that follow it. For example, given a small corpus of fifteen words as shown in Figure 3.5, the successor variety of a prefix **(r)** is **3**, since this prefix is followed by {e, i, o}; for the prefix **(rea)** it is **1**. Table 3.2 shows a complete successor varieties for all the substrings in the English word **READABLE**, based on the corpus presented in Figure 3.5.

In order to determine the stem for a given word, Hafer et al. [71] suggest to use one from four basic segmentation strategies, which are:

1- CUTOFF: In this segmentation strategy, some cutoff value is selected for

ABLE	APE	BEATABLE	FLXABLE	READ
READING	READS	RED	ROPE	RIPE
READABLE				

Figure 3.5: An Example of Small Corpus used for Successor Variety Stemmer

Step	Prefix	Successor Variety	Followed Characters
1.	R	3	E, I, O
2.	RE	2	A, D
3.	REA	1	D
4.	READ	**3**	A, I, S
5.	READA	1	B
6.	READAB	1	L
7.	READABL	1	E
8.	READABLE	1	BLANK

Table 3.2: Successor Varieties For the Word **READABLE** [71]

successor varieties to identify stems. Despite this is a simple method, the problem is how to select the optimal cutoff value.

2- PEAK AND PLATEAU: In this strategy, the segmentation is done at the prefix whose successor variety is greater than both its preceding and following prefixes. Referring back to Table 3.2, it is noted that the value of successor variety decreases as the length of prefix increases. The exception to this is at the morpheme boundary. The successor variety of the morpheme in a given word is greater than the successor variety of the strings that precede and follow it. For example the word **READABLE** could be stemmed to **READ**, since, the successor variety of this stem is **3**, which is greater than that of **REA** and **READA**.

3- COMPLETE WORD: In this strategy, the segmentation is made if the segment is a complete word in the dictionary, such as **READ** in Table 3.2.

4- ENTROPY: This strategy is based on the distribution of successor variety letters to determine a set of entropy measures of word segments. The entropy

value is calculated as shown in Eguation 3.3.

$$H_{\alpha i} = -\frac{|D_{\alpha ij}|}{|D_{\alpha i}|} * log_2 \frac{|D_{\alpha ij}|}{|D_{\alpha i}|} \tag{3.3}$$

where, $|D_{\alpha i}|$ is the number of the words in the corpus that are beginning with the prefix α, and $|D_{\alpha ij}|$ is the number of the words in $D_{\alpha i}$ having successor j.

Hafer et al. [71] used various combinations of the segmentation strategies to perform **15** comparative experiments with two English corpora. They reported that successor counts stabilised when the corpus reached about **2,000** words. To select the stems after segmentation, they suggested the following rule:

if (*first segment occurs in* $\leq$ 12 *word in corpus*) *then*
    first segment is stem
else
    second segment is stem;

In conclusion, the successor variety is an independent language technique, it requiring no suffix list or removal rules. Furthermore, it could be applied to different languages without any modifications. But the disadvantage of this strategy it requires a predefined corpus, and needs a time to segment the given word.

3.3.4 Affix Removal Strategy

Affix removal is considered as the most common stemming strategy for English language. It uses morphological rules to extract common stems by striping off the suffixes and/or prefixes from words [67]. In some cases, the resultant stems may need to be transformed. The affix removal usually uses iterative steps in order to remove the longest possible affixes from a word according to a special set of rules and affix lists.

A simple example of an affix removal stemmer is the **S-stemmer** [75]. This

stemmer conflates the plural word forms to singular forms [75]. Figure 3.6 shows the rules for a version of **S-stemmer**.

IF a word ends in **ies** but not **eies** or **aies**
THEN **ies** → **y**

IF a word ends in **es** but not **aes**, **ees**, or **oes**
THEN **es** → **e**

IF a word ends in **s** but not **us** or **ss**
THEN **s** → **NULL**

Figure 3.6: The S-stemmer

For example, according to rule 3 of the above stemmer, if any word ends in letter **s**, this letter should be removed, and leave the rest of the word unchanged (**books → book**).

The main advantage of affix removal stemmers is that thay are easy to implement and achieve good results, depending on the effort of implementation. But, on the other hand, these stemmers are generally designed for each specific language, and need some linguistic expertise in the language when compiling the set of rules.

In literature, many affix removal stemmers for English language are available. The most commonly used [110] are those that were developed by Porter [120], Paice/Husk [113] and Krovetz [95]. The coming pages deal with presenting these three stemmers in addition to the Lovins [100] stemmer which is the first published algorithm.

1- Lovins Algorithm

The Lovins stemming algorithm [100] is regarded as the first published stemming algorithm. It was developed by **Julie Beth Lovins** of Massachusetts Institute of Technology (MIT) in 1968 as a part of an information retrieval

system [134]. This algorithm is used as a main component of the INTREX project On-Line retrieval of library type material [112].

The Lovins algorithm is based on using a list of English suffixes. The list has **294** suffixes, where each suffix is associated with one of the **29** context-sensitive condition rules. The suffix list is subdivided into **11** sublists according to the length of these suffixes. Lovins stored these lists in decreasing order according to the suffix lengths, while each list is alphabetically sorted. The Lovins algorithm stemmed a given word in just a single pass using two main steps. In the first step, the longest match suffix found in the suffix lists that satisfied its associated condition rule is removed, while, in the second step, the word endings are checked against **35** transformation rules to minimise the chance of erroneous stripping. The complete lists of suffixes, condition rules, and transformation rules could be found in [3].

2- PORTER ALGORITHM

The Porter algorithm [120] is the most commonly cited stemming algorithm which is regarded as one of the most complete stemmers in the field. It was developed by **Martin Porter** at Cambridge University in 1980.

The Porter stemming algorithm is similar to the Lovins [100] one. it tries to reduce the number of steps that is in the Lovins stemmer. The basic idea of the Porter stemmer is that it uses the iterative steps to remove the longest possible suffixes rather than using one step. It removes about **60** suffixes in these multi-steps. The stemmer views suffixes as a combination of simpler suffixes.

The suffix stripping process is performed in five main steps. At each step, a certain suffix is removed by means of applying a predefined rule. In each step, a rule is tested. If its condition is held, then the suffix is stripped off and the process will continue to the next step. Otherwise, the remaining rules in the steps are tested sequentially. The complete algorithm of the Porter stemmer is too long and intricate to present here, the summarisation of it is illustrated

in Algorithm 3.1, while Tables 3.3 and 3.4 presented the two parts of the first step. The complete list of all rules of the Porter stemmer is presented in [120].

ALGORITHM 3.1 : Porter Algorithm

```
1: Step1a(W)
2: Step1b(S)
3: if the 2nd or 3rd rule of Step1b was used then
4:     Step1b(S)
5: end if
6: Step1c(S)
7: Step2(S)
8: Step3(S)
9: Step4(S)
10: Step5a(S)
11: Step5b(S)
```

Conditions	Suffix	Replacement	Example
NULL	sses	ss	caresses → caress
NULL	ies	i	poinies → poini ties → tie
NULL	ss	ss	carress → carress
NULL	s	NULL	books → book

Table 3.3: Step1a Rules of the Porter Stemmer

Conditions	Suffix	Replacement	Example
(m>0)	eed	ee	feed → feed agreed → agree
(*v*)	ed	NULL	plastered → plaster bled → bled
(*v*)	ing	NULL	motoring → motor sing → sing

Table 3.4: Step1b Rules of the Porter Stemmer

3- PAICE/HUSK ALGORITHM

The Paice/Husk stemming algorithm [113] was developed by **Chris Paice** at Lancaster University in 1980, and it was implemented and tested with assistance from **Gareth Husk**. The stemmer is an iterative stemmer, based on a

set of rules listed in one table [113]. Each rule could be a deletion or a suffix replacement rule. The rules are subdivided into sub-groups (sections) according to the final letter of the suffix. According to Paice [113], the reason behind this is to speedup the access to the rules table, by looking at the final letter of the suffix of the stemmed word.

The Paice/Husk algorithm works as shown in Algorithm 3.2, which was adapted from [113].

ALGORITHM 3.2 : The Paice/Husk Algorithm

```
1: Accept an English word.
2: Build a rule index by inspecting the final letter of the word.
3: Search for a section corresponding to that letter.
4: if such section is found then
5:     Consider the first rule in the relevant section.
6: else
7:     Terminate the stemming process.
8: end if
9: Check applicability of the selected rule.
10: if the selected rule is applicable then
11:     Perform the next step.
12: else
13:     Goto Step 3.
14: end if
15: Apply the selected rule.
16: if the rule has a continuation symbol then
17:     Goto Step 21
18: else
19:     Terminate the stemming process.
20: end if
21: Look for another rule in the table.
```

For example, the Paice/Husk algorithm stemmed the English word **maximum** to the stem **maxim** using the rule **(mu*2.)**. In this rule, **(mu)** indicates that the stemmed word ends with **um** (held in the reverse order in the rule itself), the star symbol (*) indicates that the word is unstemmed before. The number **(2)** in the rule, represents that two letters should be removed. Finally, the last symbol in the rule is for the the continuation symbol (see Step 16 of Algorithm 3.2). The continuation symbol may be the symbol (.), to terminate the stemming process, or the symbol (>) to indicate that the stemming pro-

cess is not finished and another rule should be applied. In our example, the continuation symbol is (.), which means that the stemming process is finished, and the suffix **um** should be removed to produce the stem **maxim**.

4- KROVETZ ALGORITHM

The Krovetz stemming algorithm [95] was developed by **Robert Krovetz** at the University of Massachusetts in 1993, which is based on using the inflectional linguistic morphology. The stemming process starts by checking the tested word against an on-line dictionary[1]. If matched then the word is returned as a result. Otherwise, the stemmer identifies the suffixes **ing**, **es**, or **ed** in the word and replaces them with the letter **e**, and rechecks the dictionary again. If it fails, the stemmer strips off the identified suffix, and rechecks the dictionary again.

The Krovetz stemmer has thee parts of conversions: (a) the conversion of plural form to singular form, (b) the conversion of past tense to present tense, and (c) the removal of **ing** suffix.

The stemmer has an exceptional list containing **60** words. For example, the English word **suited** is stemmed to **suit**, since this word is found in the exceptional list. While the English word **suites** is stemmed to **suite** by removing the suffix **s**.

3.4 Classes of Stemming Techniques for Arabic Language

In the previous sections, we discussed the classes of stemming strategies for English language. Also, we presented the four most commonly used English stemmers. However, these stemming strategies are not fully appropriate for Arabic language. This is mainly due to the morphological and semantic differences between English and Arabic. Some of the main differences are as follows [63]:

[1]The Longman Dictionary of Contemporary English (LDOCE)

1. The structure of Arabic affixes is different as Arabic is under the Semitic type of languages while English is of Indo-European type;
2. The word formation in Arabic mainly depends on roots and patterns;
3. Arabic root consonants might be changed during the morphological process;
4. The root consonants might be deleted during the morphological process;
5. Most English stemmers deal with suffixes only, but in Arabic there are suffixes, prefixes, and infixes.

Strategies for the development of Arabic stemmers are restricted due to its complicated structure. These strategies are mainly depending on the understanding of the Arabic morphology. Hence, Arabic stemming is actually a process of morphological analysis applied to the word in order to extract the correct root or stem.

In the literature, there is no standard classification for stemming strategies for Arabic language. Table 3.5 summarised all the classifications that are found in the literature. The best classification is the one that done by Al-Sughaiyer and-Al-Kharashi [28] in 2004. Al-Sughaiyer et al. [28] classified the stemming strategies for Arabic language into four main approaches, namely, table lookup, combinatorial, linguistic, and pattern-based approach. This classification is a comprehensive and it is the latest one found in the literature.

We followed the same classification of Al-Sughaiyer et al. [28] with some modification. Al-Sughaiyer et al. [28] classified the pattern-based class as a separate group. But really, this group is a linguistic approach and depends on the language patterns. In our classification, pattern-based is a sub-class of the linguistic approach. In summary, we classified the stemming strategies for Arabic language into three main approaches, namely, table lookup, linguistic, and combinatorial approaches. Figure 3.7 shows these approaches, and the next subsections deal in detail with these approaches.

No.	Authors	Year	Classifications
1	Hilal [141]	1989	- Pattern Compatibility - Augmented Transition Networks (ATN)
2	Al-Fedaghi & Al-Anzi [15]	1989	- Linguistic Approaches - Combinatorial Approaches
3	Fraser, Xu & Weischedel [140]	2002	- Stem-based - Root-based
4	Darwish [58]	2002	- Symbolic Approach - Statistical Approach - Hybrid Approach
5	Larkey, Ballesteros & Connell [97]	2002	- Manually Constructed Dictionaries - Algorithmic Light Stemmers - Morphological Analysers - Statistical Stemmers
6	Al-Sughaiyer & Al-Kharashi [28]	2004	- Table Lookup - Combinatorial - Linguistic - Pattern-based Approach

Table 3.5: Different Classifications for Stemming Strategies for Arabic Language

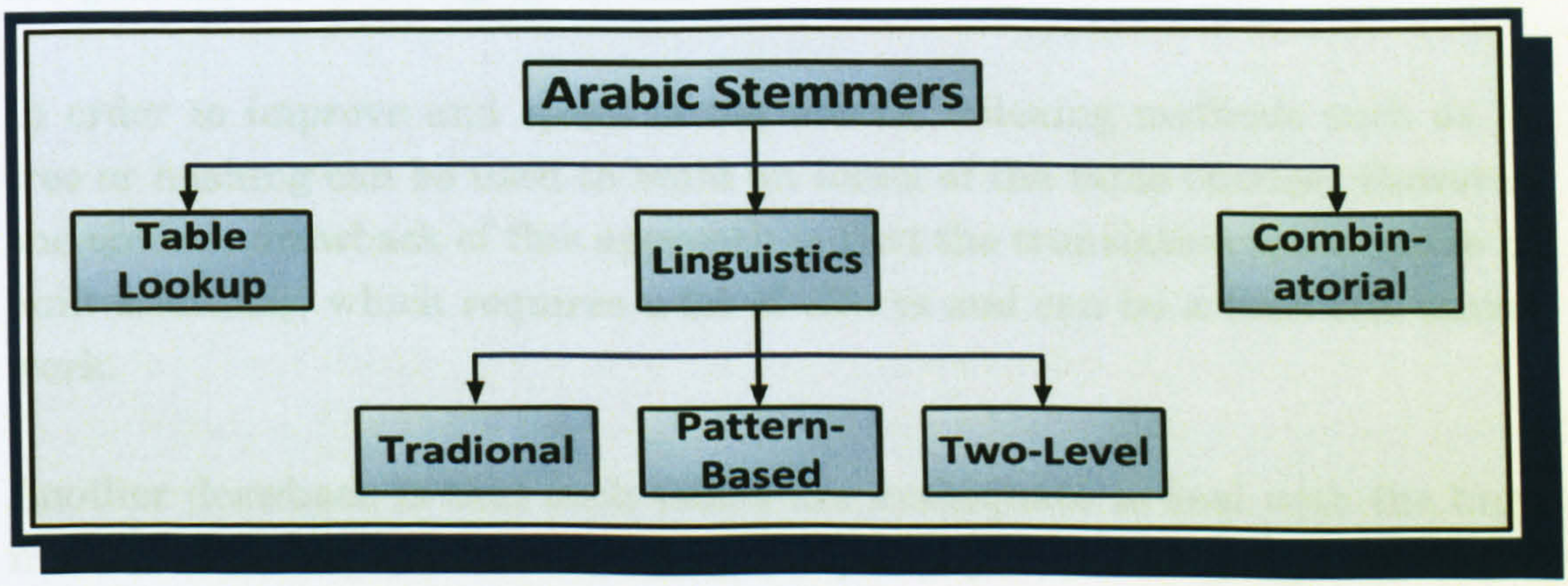


Figure 3.7: Classification of Arabic Stemmers (modified from [28])

3.4.1 Table Lookup Strategy

Table lookup stemmer strategy is the same as for the English language (see Section 3.3.1 Page 60 of this chapter). In this type of stemming, all the Arabic words are stored in a huge sized translation table of two columns. The first column is for the original word forms, while the second one is for the

corresponding their roots or stems. An example of such a table is shown in Table 3.6.

Term	Root
الكتاب	كتب
معلوماتهم	علم
بالجامعات	جمع
جوامع	جمع
مكتبات	كتب
العولمة	علم

Table 3.6: Arabic Table Lookup Stemmer Example

A given word is stemmed by accessing the lookup table and retrieving the root/stem associated with it. The stems or roots obtained using lookup tables are guaranteed to be highly accurate.

In order to improve and speed of the access, indexing methods such as B-tree or hashing can be used to build an index of the table entries. However, one obvious drawback of this approach is that the translation table has to be built manually, which requires a lot of efforts and can be a time consuming work.

Another drawback is that such tables are inadequate to deal with the huge number of words and their formation, especially in the language that has a rich morphology, such as Arabic language. According to Larkey et al. [96], table lookup approach is "obviously impractical for realistic sized corpora".

Moreover, even such a table exists, it requires significant effort in updating maintenance, to keep up with the new word formation, and this is actually not an easy task. An example of a new word that entered the Arabic dictionaries in the last view years is the word [العولمة](Al'wlmt,"Globalisation").

3.4.2 Combinatorial Strategy

Combinatorial stemming is a trial and error strategy [28], based on the combinations of letters of a given word. In this strategy, all combinations of three or four letters of a given word are compared against the roots table. If matched, the associated root is resulted; otherwise the word itself is considered as a root. Such stemmers are very simple, and easy to implement. Furthermore, the stemmers require little knowledge of Arabic morphology [15]. But they are time consuming. For example, a word with a length of **8** letters, there were **56** different combinations of three letters, a worst case all of which should be checked and for a word with a length **12** letters, there were **220** different combinations of three letters.

The most well-known combinatorial stemmers are those that are developed by: (a) Al-Fedaghi and Al-Anzi [15], (b) Al-Shalabi and Evens [127].

1- Al-Fedaghi & Al-Anzi Stemmer

Al-Fedaghi et al. [15] in 1989 presented an algorithm to generate the root and the pattern of a given Arabic word. The stemmer uses two main files: the first file contains all triliteral roots and the second contains all patterns with all possible affixes attached to them. The main idea in this algorithm is to locate the positions of the root's letters in the pattern and examine the letters in the same position in a given word to see whether they form a valid Arabic root or not. The algorithm examines all patterns that have the same length as the input word in case that word contains its full triliteral root. For those that lose one of their letters, the algorithm tests all patterns of lengths equal to one less than the length of the given word, and for the roots that lose two letters it examines all patterns of length equal to the length of the given word minus two. Then, the algorithm has to check the input word against four modules to cover all possible orthographical and phonological cases that may affect the formation of the word.

Al-Fedaghi and Al-Anzi stemmer is very simple and straightforward. But,

in contrast, it is very slow [28]. The complexity of this algorithm is measured by $O[w^3]$ whereby **w** is the word's length).

2- AL-SHALABI & EVENS STEMMER

Al-Shalabi et al. [127] in 1997 designed and implemented an Arabic stemmer based on Al-Fedaghi and Al-Anzi algorithm [15]. This stemmer covered the **quadriliteral** and **pentaliteral** roots, in addition to the **triliteral** roots. Al-Shalabi stemmer process starts by removing the longest possible prefix from the word, assuming that the first letter of what is left is a root letter. According to the authors, the root must be found in the first four or five letters of the remainder after the prefix has been removed. The possible trigrams (combinations) of these letters are generated. Then every trigram is checked against a list of known roots, and the first trigram matching the list is returned as a root. Once a root is returned, a pattern corresponding to the stemmed word conducts through interdigitisation the non-root letters with the base-pattern form. Then the resultant pattern is checked against a predefined patterns list.

Al-Shalabi et al. reported [127] that their stemmer is faster than Al-Fedaghi and Al-Anzi stemmer. The author agree with this, since the number of letter combinations in Al-Shalabi algorithm is less than that is in Al-Fedaghi algorithm. But, the comparison between the two stemmers was not provided. Al-Sughaiyer et al. stated that the success rate of Al-Shalabi stemmer based on stemming noun words is **19%** [28], which is a very low success rate. In our opinion, there are two reasons behind this: first, the algorithm does not include all Arabic patterns, but only a limited number of patterns (**39** patterns). If a word's pattern is not included in the list, then the algorithm may apply the wrong pattern and accordingly the wrong root. Secondly, the checked root file is also too small.

3.4.3 Linguistic Strategy

The linguistic strategy (also referred as rule-based strategy) is a commonly applied stemming technique. It is based on the linguistic rules obtained

through a detailed analysis of Arabic morphology system. The rules describe the morphological structure of the words [91]. The linguistic strategy simulates the same process of an expert linguist during his analysis of a given Arabic word. The linguistic strategy is subdivided into three main approaches, the traditional, the pattern-based and the two-level approaches. The next subsections deal with these approaches.

3.4.3.1 Traditional Approach

Traditional stemming approach tries to extract the root through stripping off the more common inflectional and morphological endings without uses the pattern lists. This approach tries to simulate the affix removal strategy in English language.

In literature, there are many stemming algorithms use the traditional approach such as Hilal Morphological Analyser [141], Awajan rule-based morphological Analyser [44] and Momani and Fraj Stemmer [105]. Actually, all these algorithms are similar, they are try to extract the root of the Arabic words by removing the prefixes and suffixes then applying a set of linguistic rules. In this section, the Momani and Fraj stemmer which is the recent algorithm among them is presented.

MOMANI AND FRAJ STEMMER

Momani and Fraj [105] presented in 2007 an algorithm to extract the trilieral Arabic word roots. Their stemmer is a rule-based stemmer uses many steps to produce the root for a given Aarabic word.

The Momani et al. stemmer has two lists, one list for the prefix letters contains 27 prefixes, while the other list for the suffix letters contains 24 suffixes. Also, the stemmer uses four arrays namely, word-array, stored-array, after deletion-array, and stem-array.

The stemmer tries to extract the trilieral root for a given word by remov-

ing the longest possible prefixes and suffixes. Then, it deletes any remaining letter has appear more than one time in the word-array and belongs to the set of the word letters [سألتمونيها] (sĀltmwnyhA).

In the next steps, the stemmer applies a sequence of rules in order to determine the exact root letters for the tested word. The complete rules and the pseudo code of the Momani et al. stemming algorithm is presented in [105]. A sample tracing of their algorithm is shown in Table 3.7. This tracing is adopted directly from [105].

Term	حكّامهم	تعالى	المعاملة	المسلمين
Stripped	حككام	عالى	معامل	مسلم
Word-Array	ح ك ك ا م	ع ا ل ى	م ع ا م ل	م س ل م
Stored-Array	ا ح ك ك م	ا ع ل ى	ا ع ل م م	س ل م م
After Deletion-Array	ا ح ك ك م	ا ع ل ى	ا ع ل م	س ل م
Stem-Array	ح ك م	ع ل ى	ع م ل	س م ل
Stem	حكم	على	عمل	سلم

Table 3.7: A Sample Tracing of the Momani and Fraj Algorithm [105]

The Momani et al. reported that their stemmer has an accuracy of **73%** when the stemmer was tested against **1500** words. From the author view, this accuracy is insufficient according to the size of the tested words.

3.4.3.2 Pattern-Based Approach

The pattern-based approach is based on a deep and a comprehensive analysis of Arabic morphology system. It depends on a regularity principle, which states that every root can be associated with a pattern. The principal steps of an Arabic stemmer, based on the pattern-based approach are as follows:

1. Compile two lists for prefixes and suffixes.
2. Compile a list of patterns.
3. Compile a list of roots.
4. Identify and remove prefixes and suffixes from the tested word to extract a stem.

5. Check the extracted stem against the patterns list.

6. If there is a match, then extract the root of the word and check it against the root list.

7. If the root is not found, then return the word intact.

The most common stemmers used on pattern-based approach are: **Khoja root-stemmer** [93] and **Al-Shalabi pattern-base stemmer** [128]. **Khoja root-stemmer** is presented in detail in the next section.

AL-SHALABI PATTERN-BASED STEMMER

Al-Shalabi pattern-based stemmer [128] was developed by **Riyad Al-Shalabi** of Yarmouk University in 2005. The stemmer tries to extract the root of a given unvocalised Arabic word by removing prefixes and suffixes. Then it matches the resultant word against a list of available patterns to find the suitable one and then extracts the three letters of the root by removing all infixes in that pattern. Al-Shababi [128] stated that his stemmer does not use any dictionary in order to check the correctness of the resultant root. Instead of that, he defined a set of rules to assert the decision if the letters belong to the root or not. Algorithm 3.3 presented the complete steps of Al-Shalabi pattern-based stemmer [128].

ALGORITHM 3.3 : Al-Shalabi Pattern-Based Stemmer [128]

```
1: Normalise corpus.
2: Remove the determiner [ال] (the) and its combinations from the beginning of the word.
3: Check for prefixes with duplicate letters and remove the first one.
4: if the first letter is [و] then
5:     Remove all suffixes.
6:     if CheckPrefix([و])=TRUE then
7:         Remove [و].
8:     else
9:         Goto Step 33.
10:    end if
11: else
12:    if the first letter is [ف] then
13:        Remove all suffixes.
14:    else
15:        if CheckPrefix([ف])=TRUE then
16:            Remove [ف]
17:        else
```

```
18:              Goto Step 33.
19:          end if
20:      end if
21:  end if
22:  if the first letter is [ك] or [ل] or [ب] then
23:      Remove all suffixes.
24:      if CheckPrefix([ك] or [ل] or [ب])=TRUE  then
25:          Remove it.
26:      else
27:          Goto Step 33.
28:      end if
29:  end if
30:  Remove non single letter prefixes.
31:  Remove all suffixes.
32:  Remove single letter prefixes.
33:  Match with pattern.
34:  if no match then
35:      Retun the original word and EXIT.
36:  else
37:      Normalise root.
38:  end if
```

The algorithm steps are very clear, and there is no need here to explain them, the full details of these steps are presented in [128]. The reason behind listing the complete steps of Al-Shalabi pattern-based stemmer [128] in this chapter is to show the reader an example of a recent algorithm that uses many steps and rules in order to extract the root for a given word.

Al-Shalabi stated [128] that his stemmer is faster than other stemmers in literature. We do not know how Al-Shalabi conducted this result, since there is no any evaluation provided in the work.

3.4.3.3 Two-Level Approach

The **two-level** theory model was presented by **Koskenniemi** [94] in 1983 which described natural language morphology as a relation of two distinct levels and one relation, which are they:

1. The **surface level** which is the actual representation of a word itself. In this level, the word is considered as a string of characters representing the word in its normal orthographic form. For example, in English language, the two words **writing** and **computing** are both surface rep-

resentations.

2. The **lexical level** which is the set of the morphemes of the word. In this level, the word is considered as a concatenation of the root and its part-of-speech tagging.

   Table 3.8 shows different English word examples and their lexical and surface forms.

Surface Level	Lexical Level
Books	Book + Noun + Plural
Write	Write + Verb
Plays	Play + Verb + 3PSg[2]

Table 3.8: Semantic Examples of Surface and Lexical Levels

3. The **relation** component is consist of a set of rules which map the previous two level representations to each other.

One of the best known and the most quoted Arabic morphological analyser uses the two-level theory is the one that provided by **Beesley**. In a series of studies [46][47][48], **Beesley** developed an Arabic morphological analyser and generator based on a finite state techniques. One level is for roots and patterns and the other level is for affixes. During the runtime, the system combined the roots, patterns, and affixes to produce the stems. The system has **113** rules, and **5,000** roots,

For an input word, this system generates all possible diacritical variation forms of the word. For every form, the root, the pattern, the grammatical features, and the meaning of the word are presented. Beesley Arabic morphological analyser has some drawbacks such as:

1. There was no automatic rule compiler and the rules are compiled by hand [32].

[2]3PSg : Third Personal Singular.

2. It is slow because there is no algorithm to intersect the rule transducers [32].

3. A large number of variations for a unique word. The author tested the system[3], using the Arabic word [معلمه] (m'lmh) and the analyser presented **39** different solutions.

3.5 Light-Stemmers versus Root-Stemmers

There is another classification of Arabic stemmers which is based on the desired level of analysis for information retrieval. These are: **light-stemmers** and **root-stemmers**. The main differences between the two class is the size and the type of the stripped affixes.

Light-stemmers which are also called **surface-stemmers** are try to strip off a small set of prefixes and/or suffixes of a given word to produce a **stem** without dealing with infixes [96]. According to Larkey et al. [96], the light stemming is generally an affix removal approach. For example, consider the Arabic word [المدرسون] (Almdrswn,"the teachers"). This word is a masculine, nominative and regular plural. So, if we stemmed it using a **light-stemmer**, the stem [مدرس] (mdrs,"teacher") will result. In this case the prefix [ال](Al,"the") and the masculine plural suffix [ون](wn,"them") were removed.

Recently, there are several Arabic light stemmers have been discussed in the literature. An example of them are: Larkey (light-10) [96], Buckwalter [50], Aljlayl [37], Darwish (al-stem) [59], and Chen/Gey [54]. Almost all of these stemmers are rule-based, and they differ on the size of the stripped affixes. Buckwalter morphological analyser [50] is the best known and the most quoted in the literature is presented below.

On the other hand, **root-stemmers** which are called **heavy-stemmers**, try to

[3]http://www.xrce.xerox.com/competencies/content-analysis/arabic/input/keyboard-input.html. It is preferred to use the Mozilla browser instead of the Internet Explorer browser.

strip off the prefixes, suffixes and infixes to produce the **root** of a given word. For example, the same Arabic word above, [المدرسون](Almdrswn,"the teachers"), after being stemmed used a **root-stemmer**, the root [درس](drs,"teach") is produced. In this case, the prefix [الم](Alm) and the suffix [ون](wn,"them") were removed. A second example, the Arabic word [الجامعات] (AljAm'At, "universities") is stemmed to [جمع](jm'). In this case the prefix [ال](Al), the suffix [ات](At), and the infix [ا](A) in the fourth position of the word, were removed.

It is clear from the previous examples that the task of a **root-stemmer** is more difficult than that is in a **light-stemmer**. The **Khoja Root-Stemmer** [93] is one of the most common root-stemmer in the field is presented below.

BUCKWALTER ARABIC MORPHOLOGICAL ANALYSER

The Tim Buckwalter morphological analyser [50] has been made available through the Linguistic Data Consortium (LDC) [98]. The current version of the analyser is version 2.0, which was developed in 2004. The Buckwalter morphological analyser uses three large lexicons contain **78,839** stems, **548** prefixes and **906** suffixes. It also uses a three predefined truth tables with **5,185** entries that contains morphological rules for all the possible combinations of those stems, prefixes and suffixes. These entries as follows: **2,435** entries for the prefix-stem combinations, **1,612** entries for the stem-suffix combinations, and **1,138** entries for the prefix-suffix combinations.

The analyser takes as input an Arabic words with or without diacritics and performs morphological analysis and part-of-speech tagging. Each input word is segmented into three parts segments: prefix, stem and suffix. The segmentation process is done according to the following rules:

- The prefix part could be 0 to 4 letters.
- The stem part could be 1 to infinite letters.
- The suffix part could be 0 to 6 letters.

Given an Arabic word, the analyser tries to produce all the possible morphological analyses of that word. As mentioned above, each analysis is composed of three segments: prefix, stem, and suffix. Then the analyser checks if all the three segments are found in their respective tables. The next step is to check the segments combination is acceptable or not by using a predefined truth tables. If the combination of the three segments is found in the truth tables, the word is valid and a stemming solution is presented.

In addition to the presentation of the morphological features of a tested word, the part-of-speech tag and English glossary are also presented. The author tested the Buckwalter morphological analyser[4] using the Arabic word [معلمه] (m'lmh,"his teacher") and two solutions were presented.

Actually, this analyser produces the stem rather than the root for a given word.

KHOJA ROOT-STEMMER

The Khoja stemmer [93] is one of the most common cited **root-stemming** algorithm. It was developed by **Shereen Khoja** at Lancaster University in 1999. We will hereafter refer to this algorithm as KHOJA-STEMMER . Most of the materials presented in this section was conducted through an investigation study of KHOJA-STEMMER software which was downloaded from her homepage [5].

KHOJA-STEMMER attempts to derive the root of a given word by removing prefixes and suffixes. The resultant stem is then checked against a list of patterns to produce the corresponding root. The stemmer uses a number of files which are essential during the stemming process. Table 3.9 summarises these files and shows the number of items in each file.

[4]http://students.cs.byu.edu/ jonsafar/buckwalter.html [Access on 31/03/2007].
[5]http://zeus.cs.pacificu.edu/shereen/research.htm#stemming

No.	File Name	No. of Items	No.	File Name	No. of Items
1.	Stopwords	168	10.	Punctuation	38
2.	Prefixes	11	11.	First Waw	192
3.	Suffixes	27	12.	First Yah	231
4.	Triliteral Roots	3,822	13.	Last Alef	21
5.	Triliteral Patterns	46	14.	Last Hamza	68
6.	Quadriliteral Roots	926	15.	Last Maksoura	2
7.	Define Article	5	16.	Last Yah	231
8.	Diacritics	6	17.	Mid Waw	290
9.	Duplicate	34	18.	Mid Yah	200

Table 3.9: The List of Files Used in Khoja Stemmer

KHOJA-STEMMER carries out a number of steps in order to extract the roots for a given set of words stored in a text file. These steps are summarised in Algorithm 3.4.

ALGORITHM 3.4 : Pseudo Code of KHOJA-STEMMER Algorithm

```
 1: Accept an Arabic Text.
 2: Diacritics Removal using file 8.
 3: Punctuation Removal using file 10.
 4: Stopwords Removal using file 1.
 5: Define Article Removal using file 7.
 6: Suffixes Removal using file 3.
 7: Prefixes Removal using file 2.
 8: Check the resultant stem against a patterns list using file 5.
 9: if matched then
10:     Extract the root.
11:     Check the resultant root against a roots list using file 4.
12:     if matched then
13:         Return the input word and its root (STEMMED).
14:     else
15:         Return the input word (NO STEMMED).
16:     end if
17: else
18:     Return the input word (NO STEMMED).
19: end if
```

In the first step, KHOJA-STEMMER accepts an Arabic text; then the stemmer tries to clean up this text from diacritics, numbers, and punctuations. This is done through Steps 2, and 3. The cleaned text is then stemmed word by word. In Step 4, the word is checked against a list of stopwords. If the word is found in the list, no stemming will be performed, and the processed word is marked

as a stopword and returned intact.

KHOJA-STEMMER process actually triggers if the examined word is not found in the stopwords list. It starts by removing the definite article and its equivalent after checking the initial letters of the word against the definite articles list (see Step 5). In Step 6 the stemmer tries to remove the suffixes of the given word, by comparing the final letters of the word against the suffixes list. KHOJA-STEMMER completes the stemming process by removing any prefixes attached to the word to produce the **stem**. **Khoja** compiled **11** prefixes and **27** suffixes and stored them in two files.

The resultant stem is compared against a predefined set of language patterns (Step 8). **Khoja** compiled a list of **46** patterns for this purpose. If the stem matches a specific pattern, the root corresponding to this stem is derived. KHOJA-STEMMER performs an extra step to check the correctness of the extracted root. This is done by matching this root against a roots list.

Khoja reported that the accuracy of her stemmer is **96%** [93]. More evaluations of KHOJA-STEMMER is presented in Chapter 7, when this stemmer is compared against the developed stemmer of this work (MUAIDI-STEMMER). The reason behind chossing this is that KHOJA-STEMMER showed superiority over previous works in the root extraction algorithms such as: **Al-Fedagi and Al-Anzi** [15], **Al-Shalabi and Evens** [127], and **Al-Shalabi** [128].

3.6 Conclusion of the First Part of the Chapter

From the different algorithm examples given in the previous sections, we can make the following observations about the recent Arabic stemming approaches found in the literature.

- English stemming strategies are not fully appropriate for Arabic language, due to the morphological and semantic differences between English and Arabic.

- The stemming in English and many other western European languages, is primarily a process of suffix removal [96], while in Arabic language it is a process of removing prefixes, suffixes and infixes.

- The majority of the existing Arabic stemming algorithms use a large set of rules, and many of them also refer to a lookup table of patterns and affixes. Table 3.10 summarises all the Arabic stemming algorithms that are presented in this chapter sorted according to the publishing year. The following points summaries the steps that are common to most of these stemming algorithms especially that are depend on the pattern lists.

  1. The largest prefix and suffix must be identified and removed to produce a word stem.
  2. The pattern of the word stem is then checked against all allowable patterns.
  3. If there is a match between the patterns, then the root of the word is generated and checked against a stored root list.
  4. If the root of the word is found in the root list, then the process ends by returning the root, pattern and suffix and prefix.
  5. If there is no match between the roots, different patterns will be tried.
  6. If all patterns produce no root, then different combinations of prefixes and/or suffixes will be suggested to produce different word stems.

This part of the chapter has considered the background of stemming algorithms. In the next part an introduction to artificial neural networks is provided, as this is the core approach adopted in **Phase II** of the research reported in this thesis.

Author(s)	Year	Approach	Required Lists	Ref.
Al-Fedachi & Al-Anzi	1989	Combinatioral	Roots, Patterns, Affixes	[15]
Hilal	1990	Traditional	Roots,Rules, Affixes, Primary Aricles	[141]
Al-Shalbai & Evens	1997	Combinatioral	Roots, Patterns, Prefixes,	[127]
Khoja	1999	Pattern-Based	Prefixes, Suffixes, Patterns, Roots	[93]
Beesley	2001	Two-Level	Patterns, Stems, Affixes, Rules	[48]
Arafat Awajan	2003	Traditional	Roots,Rules, Affixes	[44]
Buckwalter	2004	Light-Stemmer	Prefixes, Suffixes, Stems, Truth Tables	[50]
Al-Shalabi	2005	Pattern-Based	Prefixes, Suffixes, Patterns	[128]
Momani and Fraj	2007	Traditional	Affixes, Rules	[105]

Table 3.10: A Summarisation of the Arabic Stemming Algorithms Presented in this Chapter

3.7 Artificial Neural Networks

Neural network is a dynamic system consisting of simple processing units called neurons or nodes. It is inspired by the way human brains operate and learn from experience. The human brain is composed of millions of cells (10^{11}) called neurons which are interconnected in very complex ways [79][85]. These connections give the brain the ability to learn, memorise and apply previous experiences to our daily actions. Each cell (neuron) acts as a processing element and connects with up to 200,000 other neurons in the human brain.

Earlier work in neural network research dated back to the 1943, when McCulloch and Pitts described a simplified model of a neuron [102]. Over the years, there have been many proposals for different neural computation models [85]. However, there are some basic elements common to most of these models.

The definition of artificial neural networks (ANNs) can be given only from a point of view of a certain field of science. There are many applications of ANNs, some of them are: pattern classification, pattern matching, pattern completion, optimisation, and simulation [85]. The following definition is adopted from Jain et al. [85] in which a ANN is viewed as an adaptive machine.

DEFINITION 3.2 (ARTIFICIAL NEURAL NETWORKS)

Artificial Neural Networks which are also referred to as neural computation, network computation, connectionist models, and parallel distributed processing (PDP), are massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections between them. It resembles the brain in two respects:

- *Knowledge is acquired by the network through a learning process.*
- *Interneuron connection strengths known as synaptic weights are used to store the knowledge.[85]*

The ANN has a training rule whereby the weights of connections are adjusted on the basis of data, which is an example of the data that will be used in the applications. Training means learning the relation between the input and the expected output. During this training, the ANN develops the ability for generalisation beyond the training data. After training, the weights are not altered and the ANN makes decisions based on the strength of its weights without resorting to any other resource in decision-making.

As we mentioned early, the fundamental processing element of ANN is a neuron. Figure 3.8 shows a simple artificial neuron. Each individual neuron in the ANN accepts several inputs, and provides only one output. The inputs to the neuron can be either outputs of other neurons or can be external inputs as the case in the input neurons. For example the neuron in Figure 3.8 has a single scalar input p and a scalar bias b. The scalar input p is transmitted through a connection that multiplies its strength by the scalar weight w, to form the product wp. Then the scalar wp is added to the bias b. The bias b is like a weight, except that it has a constant input of 1. The resulting addition will be the argument of the activation function f to produce the scalar output a. The meaning of the activation function and the most commonly used activation functions is detailed in Section 3.7.2.

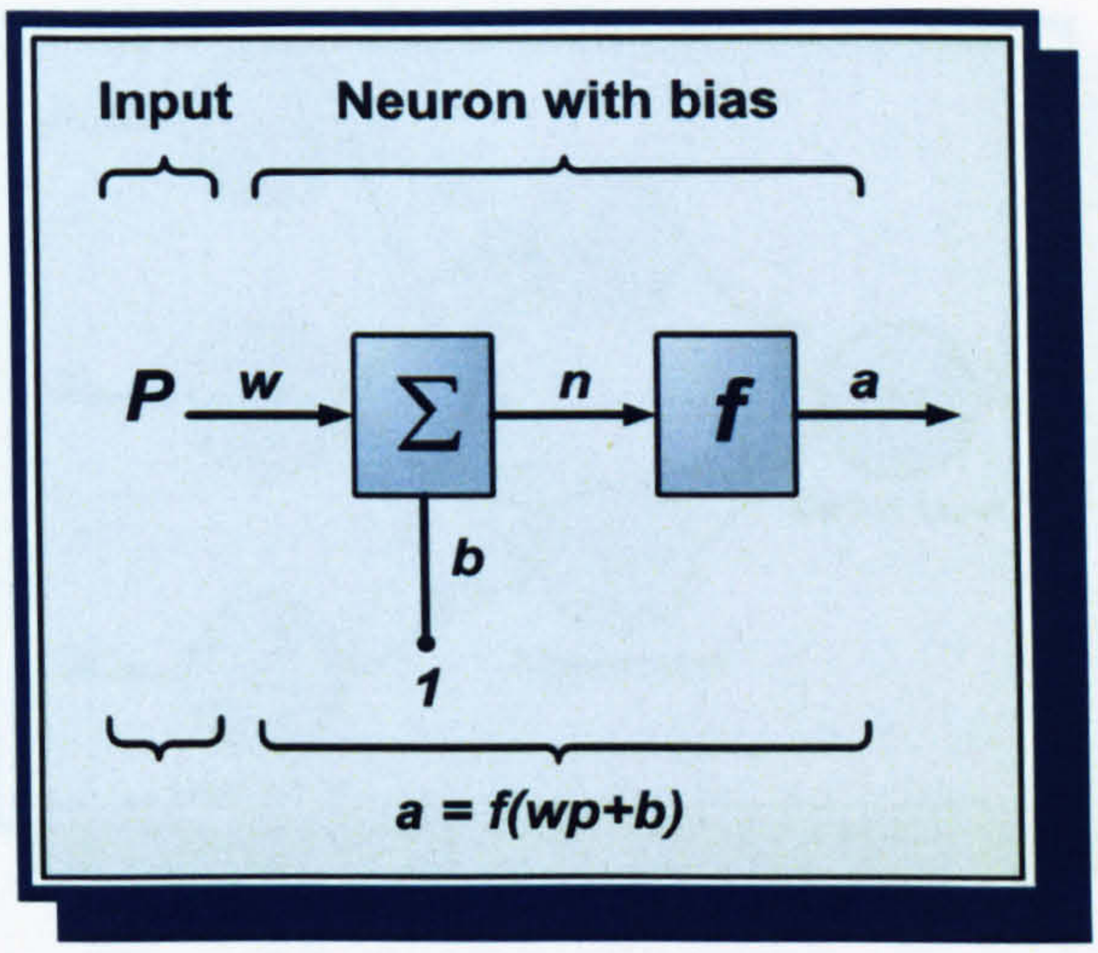


Figure 3.8: Simple Neuron Model [72]

3.7.1 Neural Network Architecture

The number of neurons, the arrangement of them into layers and the way how they are interconnected determine the so-called topology or architecture of ANN [130]. In many networks, the most common structure of connecting neurons is by layers. There are two types of architectures: (a) signal-layer network, and (b) multi-layer network. The single-layer network architecture has just two layers: the input layer and the output layer. Whereas, the multi-layer network architecture has more than two layers: the input layer, one or more hidden layers and the output layer. The presence of these hidden layers allows the solution to more complex problems, such as those that the single-layer network can not solve.

Figure 3.9 shows a multi-layer ANN with three layers of neurons. For brevity, this ANN is referred to as a 3-2-1 network. The input layer has 3 neurons, the hidden layer has 2 neurons, and the output layer has only 1 neuron. Every layer has its own neurons. The left neurons i_1, i_2, i_3 in Figure 3.9 compose the input layer. The middle neurons h_1, h_2 form the hidden layer, while the right neuron o_1 is the output layer.

The input layer neurons distribute the external inputs (x_1, x_2, x_3) to the hid-

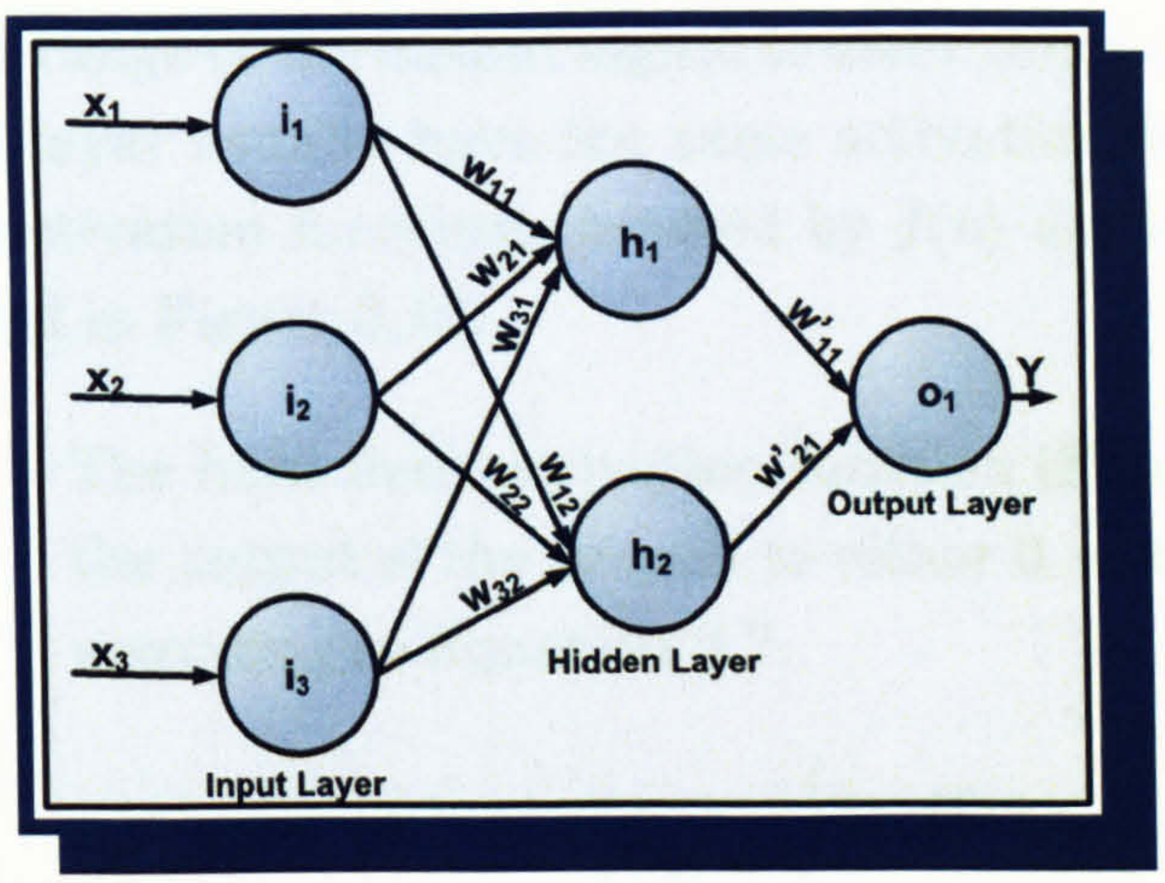


Figure 3.9: Example of a Multi-Layer ANN

den layer which performs no computation. Each of the input neurons is connected to every neuron in the hidden layer through weighted connections. This is the same for hidden neurons and the output neuron. The weights between the input neuron i_1 and the hidden neuron h_1 are $\{w_{11}, w_{21}, w_{31}\}$. The weights between the input neuron i_2 and the hidden neuron h_2 are $\{w_{12}, w_{22}, w_{32}\}$ while, the weights between the hidden neuron h_1 and h_2 and the output neuron o_1 are $\{w'_{11}\}$ and $\{w'_{21}\}$ respectively. The weights give the neural network its learning ability. When the weights are determined, then the summation function produces a net input as follows. In Equation 3.6 below the symbol **f** refers to the activation function. The purpose of this activation function and the different types of activation functions are discussed in the next subsection.

$$sum(h_1) = x_1 * w_{11} + x_2 * w_{21} + x_3 * w_{31} \tag{3.4}$$

$$sum(h_2) = x_1 * w_{12} + x_2 * w_{22} + x_3 * w_{32} \tag{3.5}$$

$$sum(o_1) = f(sum(h_1)) * w'_{11} + f(sum(h_2)) * w'_{21} \tag{3.6}$$

3.7.2 Activation Functions

The ANN activation function which is also referred to as a squashing function is the function that describes the neuron output behaviour. It squashes

or normalises the range of the output signal to some finite value. All the neurons in the same layer usually have the same activation function. The most commonly used activation functions denoted by $f(n)$ are listed below which are also illustrated in Figure 3.10:

1- HARD-LIMIT The hard-limit activation function (Figure 3.10 (a)) limits the output of the neuron to either 0 or 1 [72]. This is done according to Equation 3.7.

$$f(n) = \begin{cases} 0 & \text{if } n < 0, \\ 1 & \text{if } n \geqslant 0. \end{cases} \tag{3.7}$$

2- LINEAR The linear activation function (Figure 3.10 (b)) is used as linear approximator in linear filters [72]. The output is the same as the input. Equation 3.8 shows the linear activation function.

$$f(n) = n \tag{3.8}$$

3- LOG-SIGMOID The log-sigmoid activation function (Figure 3.10 (c)) takes the input n, where $n \in (-\infty, \infty)$, and reduces the output to the range 0 to 1. This is done according to Equation 3.9. This activation function is commonly used in backpropagation ANN [85].

$$f(n) = \frac{1}{1 + e^{-n}} \tag{3.9}$$

4- TAN-SIGMOID The tan-sigmoid or hyperbolic tangent activation function (Figure 3.10 (d)) is similar to log-sigmoid. It scales the output to between -1 and 1. This is done according to Equation 3.10.

$$f(n) = \frac{1}{1 + e^{-2n-1}} \tag{3.10}$$

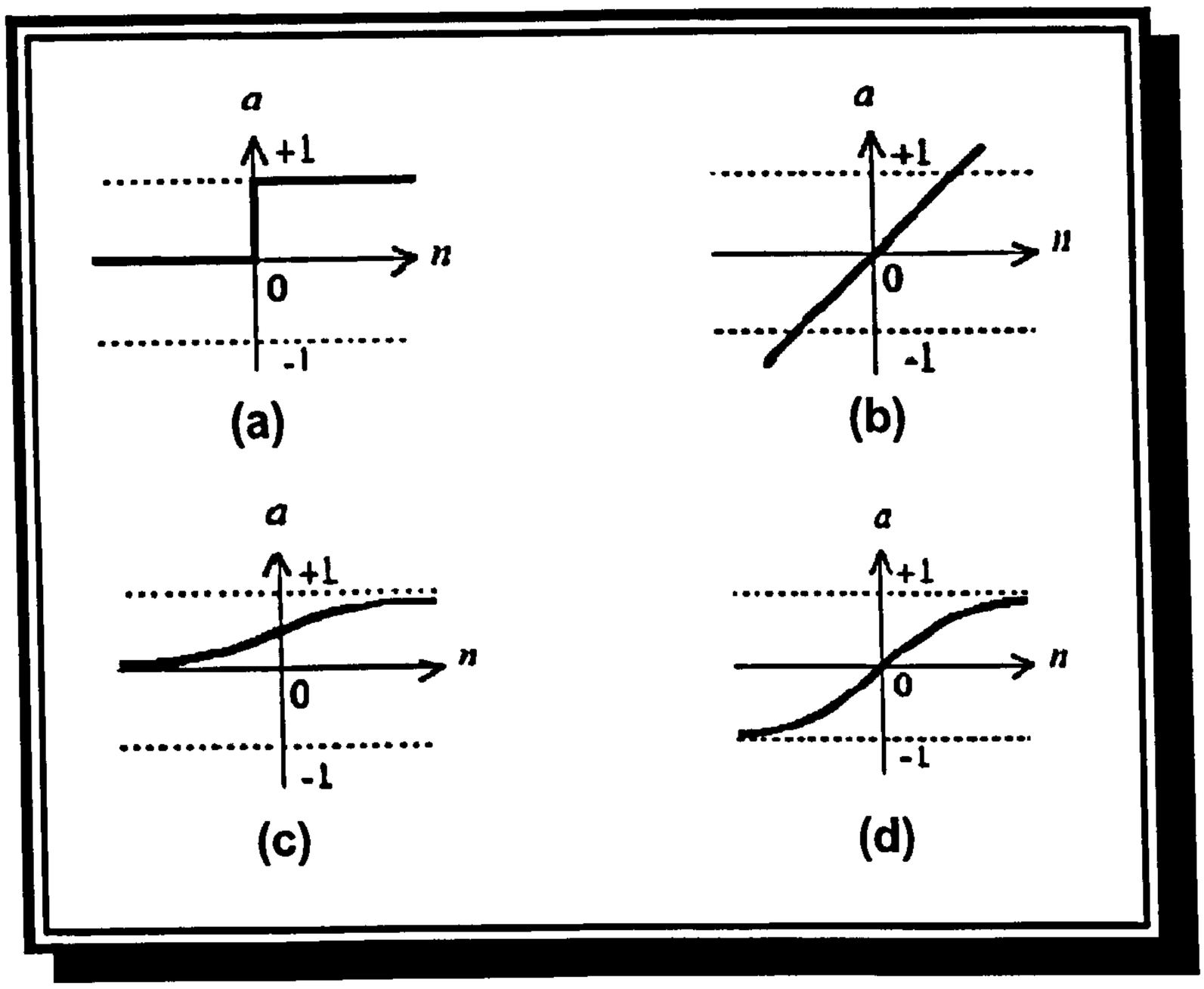


Figure 3.10: Activation Functions (a) Hard-limit (b) Linear (c) Log-Sigmoid (d) Tan-Sigmoid [72]

3.8 Network Learning

One of the most important features of ANNs is their ability to learn and generalise. Learning is achieved by training the ANN with a set of training examples. A learning process can be viewed as a problem of updating the weights so that a network can efficiently perform a specific task. In order to achieve the effect of learning, the ANN must make changes to its connection weights according to some learning rule. Depending on whether an external teacher is presented during learning, there are two different learning paradigms, namely: (a) supervised learning, and (b) unsupervised learning. The choice between these paradigms depends on the type of problem to be solved [87]. The next two subsections deal with these types of learning paradigms.

3.8.1 Supervised Learning

Supervised learning is a popular paradigm learning in ANN [76]. It refers to the learning in which an external teacher is present. In the case of ANN learning, this means that the training set is a pair formed by an input vector and the corresponding target output. In order to learn, the ANN computes the network outputs and compares its resulting outputs against the target outputs. The difference between the target output and the resulting output is called the error. The ANN tries to adjust the weights according to some learning rule in order to minimise this error. This process occurs over and over as the weights are continually modified until the global error associated with the training set reaches an acceptable value [60].

3.8.2 Unsupervised Learning

Unsupervised learning which is also referred to as a self-organisation learning, has no external teacher who is giving feedback to the network. In this paradigm, the ANN makes adjustments to itself based only on a set of input examples given to it. In other words, the training set is formed by an input vector only without any corresponding target output. One common application which often makes use of unsupervised learning in ANNs is in clustering a set of input examples. In this case, the ANN learns to discover correlations and regularities in the inputs, and adjusts its response with respect to the inputs accordingly.

In this thesis, we are only interested in the supervised learning paradigm, as it is the learning paradigm for the backpropagation ANN on which our Arabic roots extraction model is based.

3.9 Backpropagation Neural Networks

3.9.1 Backpropagation Neural Networks Architecture

There are several types of ANN architectures and training techniques that can be used to make an ANN capable of sorting classification problems. The most popular, frequently used and well studied ANN models is the backpropagation neural network (BPNN) [131][108][122], which is used approximately in 80% of all ANN applications [130].

The BPNN learning algorithm is a supervised learning algorithm for feed-forward ANNs [86]. A typical BPNN is shown in Figure 3.11. The neurons are arranged in layers and each neuron in a layer has all its inputs connected to the neurons of a preceding layer (or to the inputs from the external world in the case of the neurons in the first layer), but it does not have any connections to neurons of the same layer to which it belongs. The layers are arrayed with one succeeding the other so that there is an input layer, multiple intermediate layers and finally an output layer. Intermediate layers, that are those that have no inputs or outputs to the external world, are called hidden layers. The number of of hidden neurons in these hidden layers and their complex arrangement, largely determine the expressive power of ANN. Figure 3.11 shows a BPNN with only one hidden layer.

The BPNN training consists of two passes of computation: (a) a forward pass and (b) a backward pass, as shown in Figure 3.11. In the **forward pass**, every neuron in the hidden layer takes numeric input values from all the neurons in the input layer, then these input values are multiplied with proper weights and then summed. Here, the weights represent the connection strengths between the neurons. The output of the hidden neuron is the nonlinear transformation of the resulting sum. Similarly every neuron in the output layer takes input values from all the neurons in the hidden layer, then these values are multiplied with proper weights and then summed. The output of this neuron is the nonlinear transformation of the resulting sum.

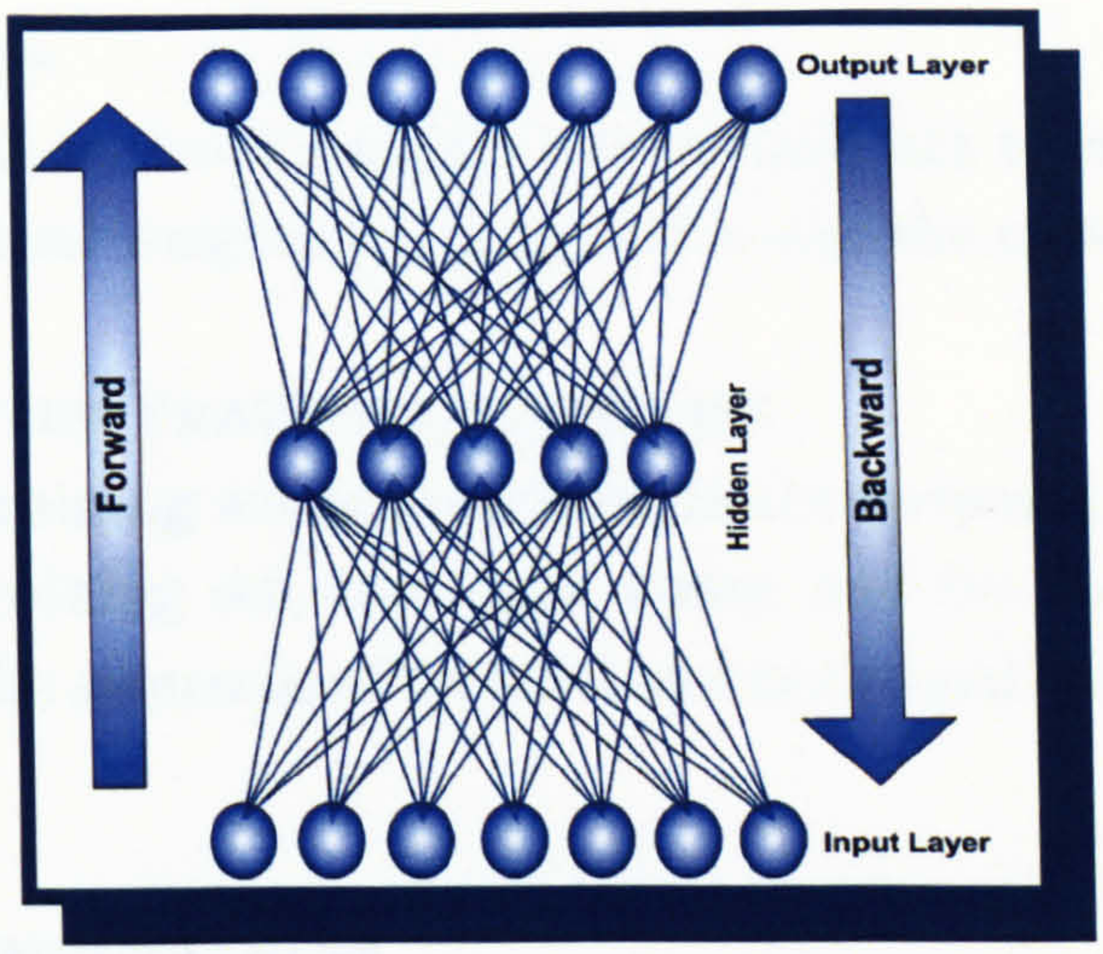


Figure 3.11: Backpropagation Neural Network

Since the resulting output and the target output are known while training the network (supervised learning), the resulting output values of the output layer are compared with the target output values. The target output values are those which should be produced by a well trained neural network. The error between resulting output values and target output values is calculated and compared. If there is no difference, no learning takes place; otherwise, this error is back propagated and utilised for adjusting the weights between input-hidden layers and between hidden-output layers. This process is called the **backward pass** of the backpropagation algorithm.

To facilitate understanding, detailed mathematical steps of BPNN algorithm are presented in the next subsection.

3.9.2 Backpropagation Neural Networks Algorithm

BPNN tries to minimise the sum of squared errors, by forcing the network weights to change in such a way that errors are minimised. The BPNN algorithm can be summarised in the following steps:

1- INITIALISATION

In this step all the network weights are initially set to small random numbers. Choosing initial weights too large will make the network untrainable.

2- PRESENTING THE TRAINING EXAMPLES

In this step the training set is presented to the network. For each training example in the training set, the input vector and the the target vector are provided. Then, the sequence of forward and backward computations are performed.

3- FORWARD COMPUTATIONS

For a given input pattern the internal activations are computed and the activation function is applied to the results. This process continues through to the output layer where the functional signals for the output neurons are computed. This in turn allows for the error signals to be computed.

4- BACKWARD COMPUTATIONS

In this step of the algorithm the differences between the resulting output and the actual output are computed and then the error is propagated sequentially backward from the output layer to the input layer, and the weights are adjusted.

The mathematical description of the BPNN algorithm is presented in Algorithm 3.5 [76]. The notations that are used based on the algorithm are presented in the Steps 1 to 10 of the algorithm.

ALGORITHM 3.5 : Pseudo Code of the Backpropagation Algorithm

1: $\vec{x}$: The vector of network input values, $\vec{x} = (x_1, x_2, \ldots, x_n)$.
2: $\vec{t}$: The vector of target values, $\vec{t} = (t_1, t_2, \ldots, t_z)$.
3: $w[\,]$: Array of weights from input layer to hidden layer.
4: $w'[\,]$: Array of weights from hidden layer to output layer.
5: $O[\,]$: Array of output of neural networks.
6: $H[\,]$: Array of hidden nodes outputs.
7: η : Learning rate parameter.
8: δ_k : The error at the output layer level.
9: δ_j : The error at the hidden layer level.
10: MSE : Mean Square Error.
11:

12: Initialise all network weights to small random numbers.
13: **while** the termination condition is not met ($MSE \leqslant 0.0001$) **do**
14: **for all** training examples ($\vec{x}$, $\vec{t}$) in the training set **do**
15: Apply the input vector $\vec{x}$ to the input units in the input layer.
16: Calculate the net input values to the hidden layer units;
17:

$$h_j = b_j + \sum_{i=1}^{n} x_i w_{ij}, \qquad (j = 1, 2, \ldots, m)$$

18: Calculate the output from the hidden layer:

$$H_j = \frac{1}{1 + e^{-h_j}}, \qquad (j = 1, 2, \ldots, m)$$

19: Calculate the net input values to the output layer units:

$$o_k = b'_k + \sum_{j=1}^{m} H_j w'_{jk}, \qquad (k = 1, 2, \ldots, z)$$

20: Calculate the outputs:

$$O_k = \frac{1}{1 + e^{-o_k}}, \qquad (k = 1, 2, \ldots, z)$$

21: Calculate the gradient terms (error terms) for the output units:

$$\delta_k = O_k(1 - O_k)(t_k - O_k), \qquad (k = 1, 2, \ldots, z)$$

22: Calculate the gradient terms (error terms) for the hidden units:

$$\delta_j = H_j(1 - H_j) \sum_{k=1}^{z} \delta_k w'_{jk}, \qquad (j = 1, 2, \ldots, m)$$

23: Update weights on the output layer:

$$w'_{jk}(new) = w'_{jk}(old) + \eta \cdot \delta_k \cdot H_j$$

24: Update weights on the hidden layer:

$$w_{ij}(new) = w_{ij}(old) + \eta \cdot \delta_j \cdot x_i$$

25: **end for**
26: Compute the mean square error (MSE):

$$MSE = \frac{1}{2} \sum_{j} (t_j - O_j)^2$$

27: end while

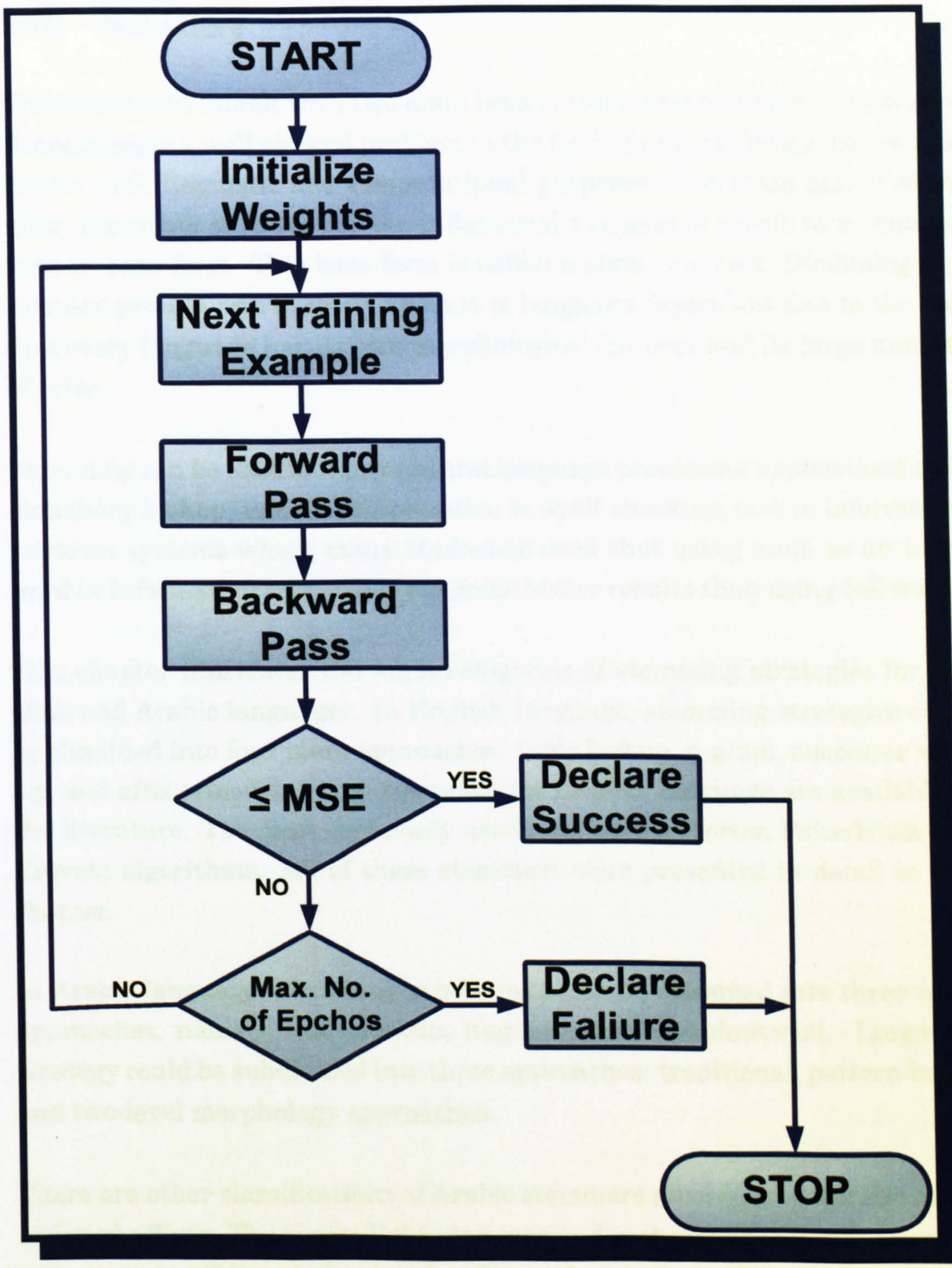


Figure 3.12: Flowchart of Training Backpropagation Neural Network

3.10 Summary

This chapter has dealt with two main issues, stemming and neural networks. **Stemming** is a well studied problem in the field of natural language processing for both linguistic and computational purposes. The main aim of stemming algorithm is to reduce the inflectional variants of words to a common form or base form. This base form is called a stem or a root. Stemming is a complex process and in many cases it is language dependent due to the fact that every language has its own morphological features and its large number of rules.

Stemming can be used in many natural language processing applications as in dictionary lookup, in data compression, in spell checking, and in information retrieval systems where many studies showed that using roots as an index word in information retrieval gives much better results than using full words.

This chapter illustrates the main categories of stemming strategies for English and Arabic languages. In English language, stemming strategies could be classified into four main approaches: table lookup, n-gram, successor variety, and affix removal. Many stemmers for English language are available in the literature. The most commonly used are: Lovins, Porter, Paice/Husk and Krovetz algorithms. All of these stemmers were presented in detail in this chapter.

In Arabic language, stemming strategies could be classified into three main approaches, namely, table lookup, linguistic and combinatorial. Linguistic strategy could be subdivided into three approaches: traditional, pattern-based, and two-level morphology approaches.

There are other classifications of Arabic stemmers according to the size of the stripped affixes. These are: light-stemmer and root-stemmer. Light-stemmer tries to strip off the prefixes and suffixes of a given word to produce a stem without dealing with infixes while, root-stemmer tries to strip off the prefixes, suffixes and infixes to produce the root of a given word. One of the most com-

monly cited root-stemming algorithm in the field is the one that was developed by Shereen Khoja. This stemmer was presented in detail in this chapter.

Neural networks are dynamic systems consisting of simple processing units called neurons or nodes. It is inspired by the way human brains operate and learn from experience. There are many functions in many fields, ANNs are able to perform, such as, pattern classification, pattern matching, pattern completion, optimisation, and simulation.

The number of neurons, the arrangement of them into layers and the way how they are interconnected determine the ANN architecture. In many networks, the most common structure of connecting neurons is by layers. There are two types of architectures: (a) single-layer network, and (b) multi-layer network. The single-layer network architecture has just two layers: the input layer and the output layer. Whereas, the multi-layer network architecture has more than two layers: the input layer, one or more hidden layers and the output layer. The presence of these hidden layers allows the solution to more complex problems, such as those that the single-layer network can not solve.

One of the most important features of ANNs is their ability to learn and generalise. Learning is achieved by training the ANN with a set of training examples. There are two major learning paradigms depending on whether an external teacher is presented during learning. These are supervised learning and unsupervised learning.

There are several types of ANN architectures and training techniques that can be used to make an ANN capable of sorting classification problems. The most popular, frequently used and well studied ANN models is backpropagation neural network (BPNN), which is used approximately in 80% of all ANN applications. The pseudo code of the BPNN Algorithm and its flowchart were presented at the end of this chapter.

The next chapter discusses the linguistic analysis of Arabic language: roots, patterns, and affixes.

Chapter 4

LINGUISTIC ANALYSIS OF ARABIC LANGUAGE: ROOTS, PATTERNS, AND AFFIXES

4.1 Introduction

Linguistically, word formation is a function of morphology [106]. As was pointed in Section 2.4 of Chapter 2, morphological analysis of Arabic language is based on roots, patterns, and affixes. One of the interesting features of Arabic language is that most of its words are generated from a finite set of roots which are interdigitised by a finite set of patterns and a finite set of affixes. The patterns are just templates with indications where the root letters should be. So, the generated words have two types of letters: root-letters and non root-letters or affix letters. Due to these interesting features, this study tries to identify the root-letters of a given word by exploiting the numerical relations between the word letters.

The chapter starts by highlighting the definition of the term root, followed by a description of the statistical analysis of the Arabic word roots based on four famous Arabic dictionaries. Furthermore, it discusses the analysis of

Arabic patterns and affixes. Finally, it deals with the process of Arabic word formation. Chapters 5 and 6 provide new computationally based approach to root extraction exploiting the analysis reported in this chapter.

4.2 Arabic Root System

The roots in Arabic language are the forms of the verb [106], while in English language a root may be a verb, a noun or an adjective. The root in Arabic language is usually found in the third person singular masculine of the past tense. In literature, there are many definitions for a root. Ryding [124] was defined the term **root** in Arabic language as:

DEFINITION 4.1 (ARABIC ROOT)

"A root is a relatively invariable discontinuous bound morpheme, represented by two to five phonemes, typically three consonants in a certain order, which interlocks with a pattern to form a stem and which has lexical meaning". [124] Page 47.

Definition 4.1 highlights some properties of the Arabic word root, which are they:

- It is a portion of the word that has two to five letters.
- It cannot be further analysable into meaningful elements (morphemes).
- It can stand on its own without the need for additional morphological elements.
- It is a discontinuous bound morpheme in which it can accept additional morphological elements to create new words.

For example, the Arabic root morpheme [كتب](ktb,"write") is a complete word with a meaningful semantic representation. It has three consonants [ك](k),

[ت](t), and [ب](b). This word root cannot be broken down to generate new words like [كت](kt) or [تب](tb). It is "discontinuous" because vowels can be inserted between the root-letters to derive new words communicate the main idea or the lexical meaning of the root. For example, if a letter [ا](A) is inserted in the middle of the current root, then a new word [كاتب] (kAtb,"writer") is generated which shares the root in the "writing" meaning. Furthermore, a letter [م](m) can be added at the beginning of [كتب](ktb) to obtain [مكتب] (mktb,"office"), and a letter [ه](h) at the end to obtain [كتبه] (kutubhu,"his books"). The previous generated words are called stems. Also, the added letters are called affixes. More details for these terms are coming in the following sections.

4.2.1 Classification of Roots According to their Lengths

The grammatical system of the Arabic language is based on a **root-and-pattern** structure and is therefore considered as a **root-based** language [90] [25]. The root contains the important part of the meaning of the word. The most commonly number of consonants in an Arabic root is three (**triliteral**) and these comprise the largest part of the Arabic language [124][15]. In addition to this, there are also two-consonant (**biliteral**), four-consonant (**quadriliteral**), and five-consonant roots (**pentaliteral**) [124]. Table 4.1 shows three different root examples for each type and their translations.

Root Type	Arabic Root	Transliteration	Translation
Biliteral	حك	Hk	to scratch
	حط	HT	put
	دس	ds	insert
Triliteral	علم	'lm	know
	درس	drs	study
	كتب	ktb	write
Quadriliteral	ترجم	trjm	translate
	سيطر	syTr	control
	دحرج	dhrj	roll
Pentaliteral	جحمرش	jHmrŝ	for old women
	سفرجل	sfrjl	quince
	كرطعب	krT'b	for small thing

Table 4.1: Different Arabic Root Examples

In literature there is no agreement on the exact number of roots in Arabic language. Darwish et al. [58] stated that the number of Arabic language roots is around **10,000** roots. Thalouth et al. [138] stated that **64%** of the Arabic roots are triliteral, **33%** are quadriliteral, and the remaining **3%** are biliteral and pentaliteral roots. Gheith et al. [68] (cited by [15]) stated that **85%** of the Arabic roots are triliteral, while Moukdad [106] and Al-Fedaghi et al. [16] stated that the number of Arabic language roots is **8,600** divided into: **6,350** are trilitrals and **2,500** are quadriliterals.

When the author started the work in this study, it was believed that it is easy to find an electronic resources for Arabic dictionaries[1]. This belief came from that a research in Arabic natural language processing was started before two decades (by the early-eighties). Unfortunately, such dictionaries are not available.

The extraction of the Arabic roots is the main core of this thesis. The resultant roots from the developed system should be evaluated by matching them against reliable Arabic dictionaries. Furthermore, such dictionaries are needed when building the training datasets that are used to train the neural networks (see Chapter 6). For these reasons, the author created a comprehensive database of Arabic roots using four standard Arabic dictionaries [106][142][5]. These dictionaries are:

1. [مختار الصحاح] Mukhtar Al-Sahah [21].
2. [العين] Al-Ayen [14].
3. [تاج العروس] Taj Al-Aroos [33].
4. [لسان العرب] Lisan Al-Arab [83].

The dictionaries creation process was done by implementing a small program. In this program we entered all the Arabic roots which are found in these dictionaries by hand[2]. Figure 4.1 shows four snapshots from these dictionaries.

[1]Referred in Arabic as Mujames.

[2]The author indebted to his wife, son, and daughter for their great help in this process.

The roots in these snapshots are shaded and circled. It is clear from these snapshots that the dictionaries are sorted alphabetically based on the word root rather than the word itself. The reason behind this is mainly due to the rich derivational features that characterise the Arabic language. Behind the root there are a lot of information to describe the root meaning and some of its derived words. In this study we are interested in the roots themselves rather than any extra information.

The total number of roots in the four mentioned dictionaries amounted to **27,457**. Some of these roots overlap. Table 4.2 shows the total number of roots in the four created dictionaries.

Mujam / Dictionary			Total No. of Roots
1-	مختار الصحاح	Mukhtar Al-Sahah	3,452
2-	العين	Al-Ayen	5,612
3-	تاج العروس	Taj Al-Aroos	9,080
4-	لسان العرب	Lisan Al-Arab	9,313
		Total	**27,457**

Table 4.2: The Number of Roots in the Four Arabic Dictionaries

Also, the same mentioned program was used to analyse the four compiled dictionaries. The number of lexical roots and root frequencies are conducted. The results of the analysis are shown in Tables 4.3, 4.4, 4.5, and 4.6 respectively. The tables clearly show that the triliteral roots are the overwhelming majority of Arabic roots, and to a lesser extent, bilateral, quadrilateral, and pentaliteral. Therefore, this thesis is concentrated upon the triliteral roots only.

Roots Type	No. of Roots	Ratio %
Biliteral	19	0.55%
Triliteral	3,239	93.83%
Quadriliteral	184	5.33%
Pentaliteral	10	0.29%
Total	**3,452**	**100.00%**

Table 4.3: Analysis of Mukhtar Al-Sahah Dictionary According to the Root Lengths

Roots Type	No. of Roots	Ratio %
Biliteral	411	7.32%
Triliteral	4,259	75.89%
Quadriliteral	881	15.70%
Pentaliteral	61	1.09%
Total	**5,612**	**100.00%**

Table 4.4: Analysis of Al-Ayen Dictionary According to the Root Lengths

مدح : المَدْح: مقيض الهجاء وهو حُسْن الثناء. والمِدْحة اسم المديح، وجمعه مَدائحُ ومِدَحٌ، يقال: مَدَحْتُه وامْتَدَحْتُه.

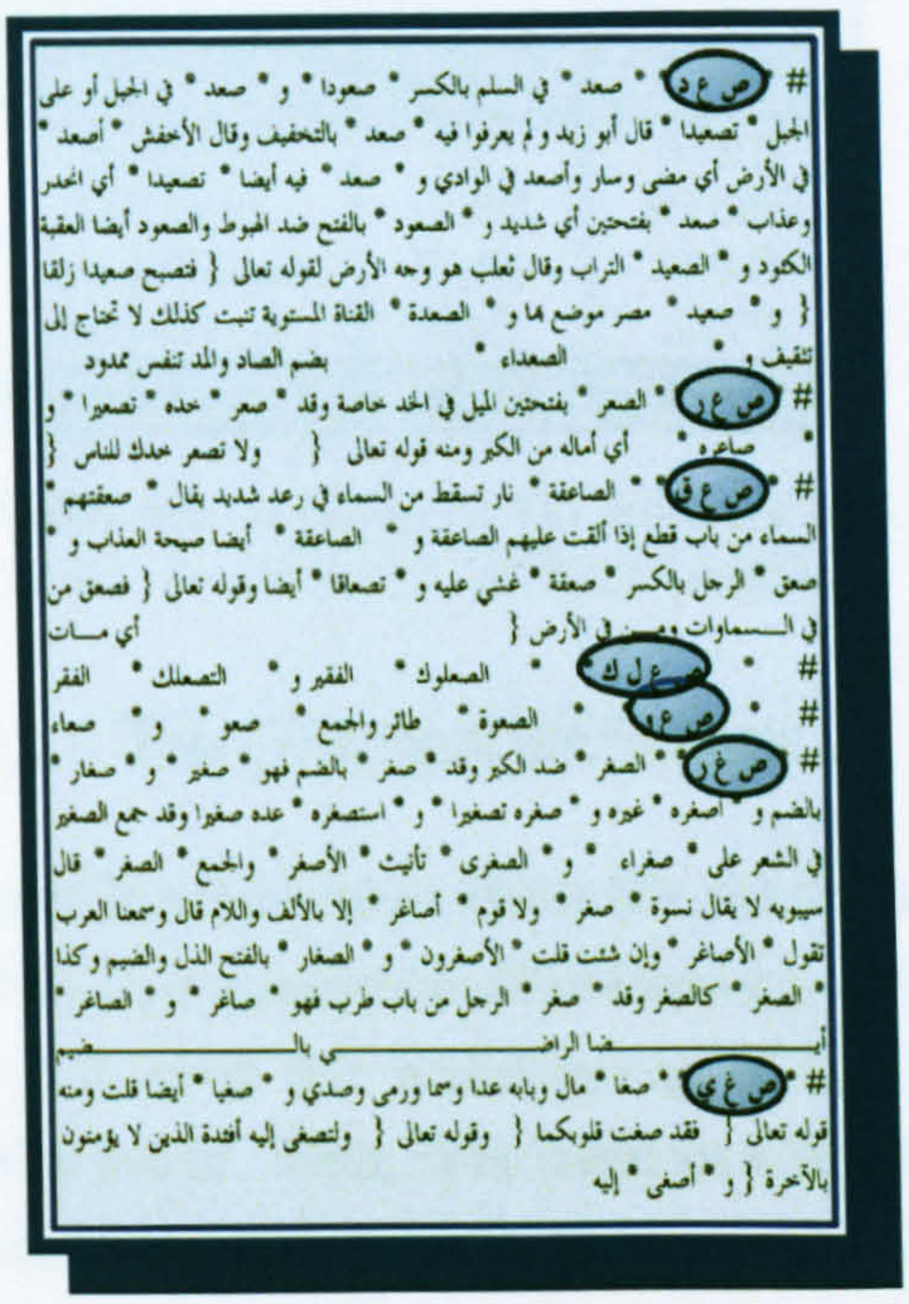
حمد: الحَمْدُ: نقيض الذَّمّ، يقال: بَلَوته فأَحْمَدْتُه أَيْ وَجَدْتُه حَميداً محموداً على الفِعْل. وحَمِدْتُه على ذلك، ومنه المَحْمَدة. وحُماداكَ أَنْ تَفعل كذا أي: حَمْدُكَ، وحُماداكَ أَنْ تنجُو من فُلان رأْساً برأْس. والتَّحميد: كَثْرة حَمْد الله بِحُسْن المَحامد. وأَحْمَدَ الرجلُ: أي: فَعَلَ فِعْلا يُحْمَدُ عليه، قال الأَعشى:

وأَحْمَدْتَ إِذْ نَجَّيْتَ بالأَمْسِ صِرْمَةً لها غَدَداتٌ واللَّواحِقُ تَلْحَقُ

والحَمْدُ: الثناء.

وخمسة من الأَنبياء ذوو اسمَيْن: أَحمدُ ومُحَمَّد -صلى الله عليه وعلى آله وسلم- وعيسى والمسيح، وذو الكِفْل وإِلياس، وإِسرائيل ويعقوب، ويونُس وذو النُّون - عليهم السلام وعلى غيرهم من أَنبيائه-. وقولهم: أَحْمَدُ إِليكَ اللهَ أي: معكَ، ويقال: إِنما هو كقولك: أَشكُو إِليك. وقوله: إِنِّي أَحمَدُ إِليكم غَسْلَ الإِحليل، أَيْ أَرْضَى لكم ذلك.

(b) Al-Ayen


ص ع د * صعد * في السلم بالكسر * صعودا * و * صعد * في الجبل أو على الجبل * تصعيدا * قال أبو زيد و لم يعرفوا فيه * صعد * بالتخفيف وقال الأخفش * أصعد * في الأرض أي مضى وسار وأصعد في الوادي و * صعد * فيه أيضا * تصعيدا * أي انحدر وعذاب * صعد * بفتحتين أي شديد و * الصعود * بالفتح ضد الهبوط والصعود أيضا العقبة الكئود و * الصعيد * التراب وقال ثعلب هو وجه الأرض لقوله تعالى { فتصبح صعيدا زلقا } و * صعيد * مصر موضع بها و * الصعدة * القناة المستوية تنبت كذلك لا تحتاج إلى تثقيف و * الصعداء * بضم الصاد والمد تنفس ممدود

ص ع ر * الصعر * بفتحتين الميل في الخد خاصة وقد * صعر * خده * تصعيرا * و * صاعره * أي أماله من الكبر ومنه قوله تعالى { ولا تصعر خدك للناس }

ص ع ق * الصاعقة * نار تسقط من السماء في رعد شديد يقال * صعقتهم * السماء من باب قطع إذا ألقت عليهم الصاعقة و * الصاعقة * أيضا صيحة العذاب و * صعق * الرجل بالكسر * صعقة * غشي عليه و * تصعاقا * أيضا وقوله تعالى { فصعق من في السماوات ومن في الأرض } أي مات

ص ع ل ك * الصعلوك * الفقير و * التصعلك * الفقر

ص ع و * الصعوة * طائر والجمع * صعو * و * صعاء

ص غ ر * الصغر * ضد الكبر وقد * صغر * بالضم فهو * صغير * و * صغار * بالضم و * أصغره * غيره و * صغره تصغيرا * و * استصغره * عده صغيرا وقد جمع الصغير في الشعر على * صغراء * و * الصغرى * تأنيث * الأصغر * والجمع * الصغر * قال سيبويه لا يقال نسوة * صغر * ولا قوم * أصاغر * إلا بالألف واللام قال وسمعنا العرب تقول * الأصاغر * وإن شئت قلت * الأصغرون * و * الصغار * بالفتح الذل والضيم وكذا * الصغر * كالصغر وقد * صغر * الرجل من باب طرب فهو * صاغر * و * الصاغر * أيضا الراضي بالضيم

ص غ ي * صغا * مال وبابه عدا وسما ورمى وصدي و * صغيا * أيضا قلت ومنه قوله تعالى { فقد صغت قلوبكما } وقوله تعالى { ولتصغى إليه أفئدة الذين لا يؤمنون بالآخرة } و * أصغى * إليه

(a) Mukhtar Al-Sahah

دخنس : الدَّخَنَّسُ: الشديد من الناس والإبل؛ وأَنشد:

وقَرَّبوا كلَّ حُلالٍ دَخَنَّسِ، عند القِرَى، حُنادِفٍ عَحَنَّسِ، تَرى على هامَتِه
كالبُرْنُسِ

درس : دَرَسَ الشيءُ والرَّسْمُ يَدْرُسُ دُرُوساً: عفا. ودَرَسَته الريح، يتعدَّى ولا يتعدَّى، ودَرَسه القوم: عَفَوْا أَثره. والدَّرْسُ: أَثر الدِّراس. وقال أَبو الهيثم: دَرَسَ الأَثَرُ يَدْرُسُ دُروساً ودَرَسَته الريحُ تَدْرُسُه دَرْساً أَي محَتْه؛ ومن ذلك دَرَسْتُ الثوبَ أَدْرُسُه دَرْساً، فهو مَدْرُوسٌ ودَرِيسٌ، أَي أَخْلَقْته . ومنه قيل للثوب الخَلَق: دَرِيس، وكذلك قالوا: دَرَسَ البعيرُ إِذا حَرِبَ حَرَباً شديداً فَقُطِرَ؛ قال جرير:

رَكِبَتْ نَوارُكُمُ بعيراً دارساً، في السُّوق، أَفْصَحَ راكبٍ وبَعِيرِ

والدَّرْسُ: الطريق الخفيُّ. ودَرَسَ الثوبُ دَرْساً أَي أَخْلَقَ؛ وفي قصيد كعب بن زهير: مُطَّرَحُ البَزِّ والدِّرْسان مَأْكُولُ

الدِّرْسانُ: الخَلَقانِ من الثياب، واحدها درْسٌ. وقد يقع على السيف والدرع والمِغْفَر. والدَّرْسُ والدَّرْسُ والدَّرِيسُ، كله: الثوب الخَلَقُ، والجمع أَدْراسٌ ودِرْسانٌ؛ قال المُتَنَخِّلُ:

قد حال بين دَرِيسَيْه مُؤَوَّبَةٌ، نَسْعٌ لها بِعِضاهِ الأَرضِ تَهْزِيزُ

ودِرعٌ دَرِيسٌ كذلك؛ قال:

مَضَى وَوَرِثْناهُ دَرِيسَ مُفاضَةٍ، وأَبْيَضَ هِنْدِيّاً طويلاً حَمائلُهْ

ودَرَسَ الطعامَ يَدْرُسُه: داسَه؛ يَمانِيَةٌ. ودُرِسَ الطعامُ يُدْرَسُ دِراساً إِذا دِيسَ. والدِّراسُ: الدِّياسُ، بلغة أَهل الشام، ودَرَسُوا

(d) Lisan Al-Arab

هبرد : (ثَرِيدَةٌ هِبْرِدَانَةٌ) ، أهمله الجوهريُّ ، وقال الأَزهريّ : أَ (بارِدَةٌ) ، هكذا تقوله العَرَبُ بكسرِ الأَولِ والثالث وسُكونِ الثاني ، وقيل : (مُصَعْنَبَةٌ مُسَوَّاةٌ مُلَمْلَمَةٌ) ، وهذا عن الصاغانيّ ، وكأَنّ : مِبْرِدَانة ، إِتْبَاعٌ .

هجد : (الهُجُودُ) ، بالضمّ ، (: النَّوْمُ) ، هَجَدَ القومُ هُجُوداً : نَامُوا ، والهاجِدُ : النائم ، (كالتهجُّد) ، في الصحاح : هَجَدَ ، وتَهَجَّدَ ، أَي نَامَ لَيْلاً ، وهَجَدَ وتَهَجَّدَ أَي سَهِرَ ، وهو من الأَضداد . (و) الهاجِدُ ، والهُجُود . (بالفتح : المُصَلِّي بالليل) و (ج) هُجُودٌ ، (بالضمّ) ، هو جَمْعُ هاجِد كواقِفٍ ووُقُوفٍ ، (وهُجَّدٌ) كرُكَّعٍ ، قال مُرَّةُ ابنُ شَيْبَانَ : أَلاَ هَلَكَ امْرُؤٌ قامَتْ عَلَيْه بِجَنْبِ عُنَيْزَةَ البَقَرُ الهُجُودُ وقال الحُطَيْئَةُ :

فَحَيَّاكِ وُدَّ ما هَدَاكِ لفِتْيَةٍ وخوصٍ بأَعْلَى ذِي طُوَالَةَ هُجَّدِ

(وتَهَجَّدَ : استَيْقَظَ) للصَّلاَةِ أَو غيرِهَا ، وفي التنزيل العزيز : { وَمِنَ اللَّيْلِ فَتَهَجَّدْ بِهِ نَافِلَةً لَكَ } (سورة الإسراء ، الآية : 79) أَي تَيَقَّظْ بالقُرْآن ، وهو حَثٌّ له في إِقامة صلاة الليل المَذْكورة في قوله تعالى : { قُمِ اللَّيْلَ إِلاَّ قَلِيلاً } (سورة المزمل ، الآية : 2) كذا في البصائر ، (كهَجَّدَ) تَهْجِيداً (، ضدٌّ) ، قال ابنُ الأَعرابيّ : هَجَدَ الرجُل ، إِذا صلَّى بالليل ، وهَجَدَ ، إِذا نَامَ بالليل ، وقال غيرُه : وهَجَدَ ، إِذا نَامَ ، وذلك كله في آخِرِ الليل . قال الأَزهريّ : والمعروف في كلام العَرَب أَن الهاجِدَ هو النائم (وهَجَدَ هُجُوداً إِذا نَامَ) وأَما المُتَهَجِّدُ فهو القائمُ إِلى الصلاة من النوم ، وكأَنه قيل له مُتَهَجِّدٌ لإِلقائه الهُجُودَ عن نفسه ، كما يُقال للعابد مُتَحَنِّثٌ ، لإِلقائه الحِنْثَ عن نفسه . وفي حديث يَحيى بن زكريّا عليهما السلامُ (فنَظَرَ إِلى مُتَهَجِّدِي بَيْتِ المَقْدِس) أَي المُصلِّين بالليل ، يقال : تَهَجَّدْتَ ، إِذا سَهِرْتَ ، وإِذا نِمْتَ ، وهو من الأَضداد .

(c) Taj Al-Aroos

Figure 4.1: Snapshots from the Four Arabic Dictionaries

Roots Type	No. of Roots	Ratio %
Biliteral	0	0.00%
Triliteral	5,502	60.59%
Quadriliteral	3,263	35.94%
Pentaliteral	315	3.47%
Total	**9,080**	**100.00%**

Table 4.5: Analysis of Taj Al-Aroos Dictionary According to the Root Lengths

Roots Type	No. of Roots	Ratio %
Biliteral	0	0.00%
Triliteral	6,560	70.44%
Quadriliteral	2,558	27.47%
Pentaliteral	195	2.09%
Total	**9,313**	**100.00%**

Table 4.6: Analysis of Lisan Al-Arab Dictionary According to the Root Lengths

4.2.2 The Common Dictionary

The extracted roots from the previous dictionaries are stored in one dictionary called the **common dictionary**. This **common dictionary** is used in Chapter 6 during the building-up of the annotated corpus to check the validation of the roots. Also, it is used in Chapter 7 during the evaluation process for the same reason.

The generated **common dictionary** has just two columns. The first column, for the entered root, while the second one for the associated dictionary. We used a string of four bits to represent this association. For example, the string <1001> means that the corresponding root belongs to the first and the fourth dictionaries, while it is not to the second and the third ones. The order of the dictionaries is as they are shown in Table 4.2. The total number of distinct roots in the **common dictionary** is **12,619**, where **4,437** roots of them are mentioned in one dictionary, **3,566** are mentioned in two dictionaries, **2,653** are mentioned in three dictionaries, and **1,963** are mentioned in all the four dictionaries. A snapshot of the common dictionary is shown in Figure 4.2, while Table 4.7 summarizes the roots in the **common dictionary** according to their length. It is clear from Table 4.7 that the vast majority of Arabic roots are thought to be made up of three consonants (triliteral roots).

Also, a comprehensive statistical analysis is done on the four mentioned dictionaries, as well as the common dictionary. The aim of this analysis is to study the combinations of consonants that could occur in the triliteral root forms in Arabic language. To achieve this goal, the same mentioned program

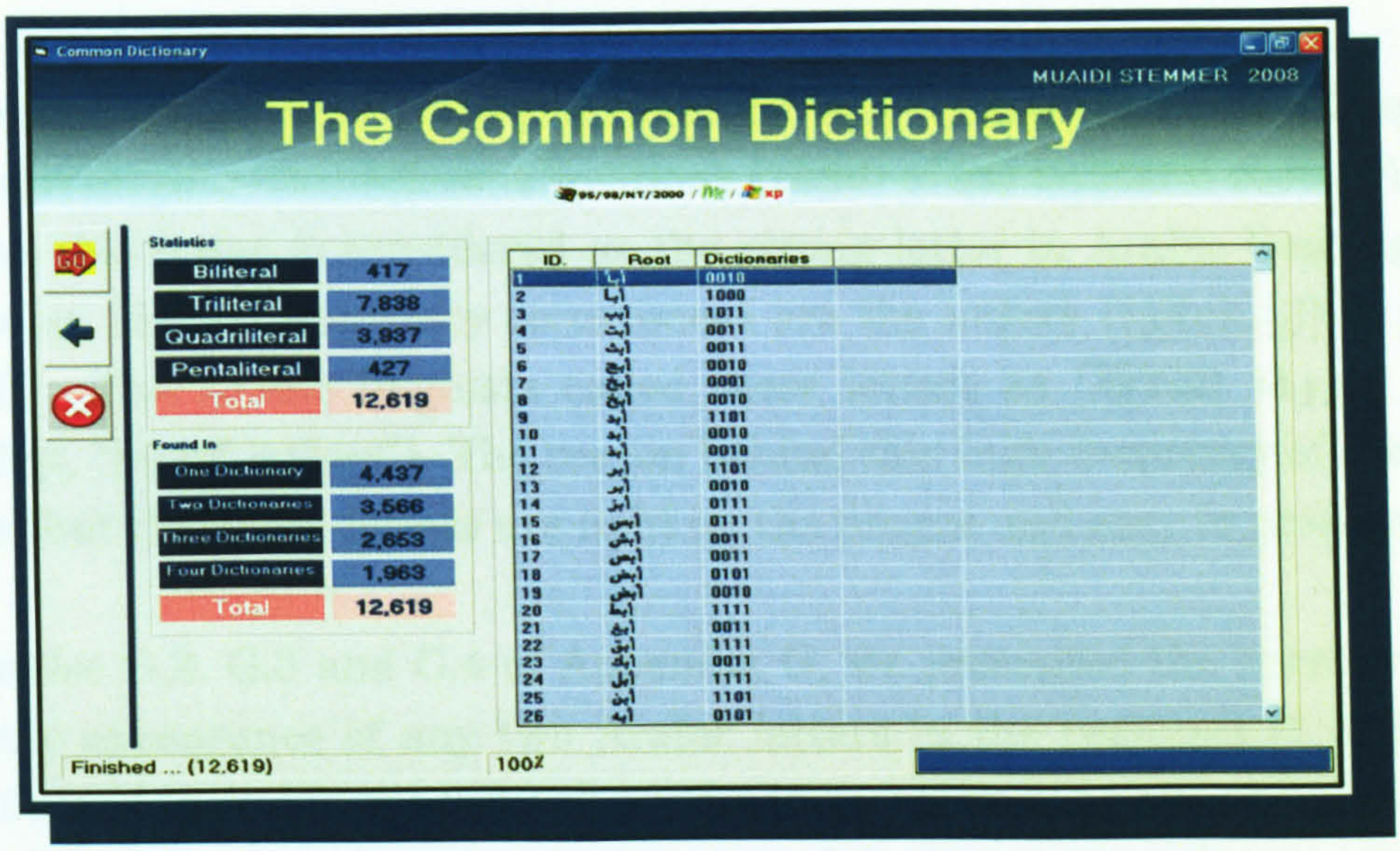


Figure 4.2: A Snapshot of the Common Dictionary

Roots Type	No. of Roots	Ratio %
Biliteral	417	3.30%
Triliteral	7,838	62.11%
Quadriliteral	3,937	31.20%
Pentaliteral	427	3.38%
Total	**12,619**	**100.00%**

Table 4.7: Analysis of The Common Dictionary According to the Root Lengths

is updated to analyse the root consonants and their combinations. In this thesis, we deal with the analysis of the **common dictionary**, since it covers all the roots from the other four dictionaries. The summary of the analysis is presented in Appendix G.

Table G.1 of Appendix G, shows the frequency of occurrences of each Arabic letter on the different positions of the triliteral roots in the **common dictionary**. For example, in **322** roots the Arabic letter [س](s) occurs in the first position of the root, in **253** roots occurs in the second position and in **293** roots occurs in the third position of the root. The frequency of occurrences of this letter in the three different positions is **868** as shown in the last column of Table G.1 of Appendix G.

It is clear from Table G.1 that the Arabic letter [ر](r) has the highest frequency of occurrences in the different positions (**1386**) of the triliteral roots. For this reason this letter is considered as the strong letter in Arabic language[17]. The next highest frequency occurrences are the letters {[ن](n), [ل](l), [م](m), [ب](b)}. The Arabic linguists called these letters as [حروف الذلاقة] (Hrwf AldlAqt, "liquid letters"). The reason behind this high frequency of these letter is that the liquid letters are light on the tongue and easy to pronounce.

In Tables G.2, G.3 and G.4 of Appendix G, we presented the number of adjacency appearance of any two Arabic letters in the positions (1 and 2), (1 and 3), and (2 and 3) respectively in the triliteral roots in the common dictionary. The following items highlight some of the interesting observations that we deduced during the study and analysis of the Arabic word roots and their positions.

- The letters [ح](H) and [غ](g) are not found in the same root.
- The letter [س](H) can not be follow by any of the following letters {[ز](z), [ش](ŝ), [ص](S), [ض](Ṡ)}.
- The letter [ص](H) can not be follow by any of the following letters {[ث](t̂), [ذ](ḋ), [س](s), [ش](ŝ), [ض](Ṡ), [ظ](t̂)}.
- The letters [ح](H) and [خ](Ĥ) are not found in the same root.

In conclusion, the **common dictionary** provides a unique set of roots for other researches in the field and is therefore a significant contribution from this work.

4.2.3 Classification of Roots According to their Types

Also, Arab grammarians classified root verbs into two major classes, according to the nature of the consonant phonemes occurring in the root: **sound verbs** and **weak verbs** [124]. The sound verbs are the verbs that only have consonant letters (i.e. do not contain any vowel letters) while, the weak verbs

have one or more vowel letters (**alef**[ا], **waaw**[و] and **yaa**[ي]). For example the Arabic word [درس](drs,"study") is a sound root verb, since it has no vowels while, the word [وجد](wjd,"find") is a weak root verb, since the first letter is a vowel. Figure 4.3 illustrates these two major classes of Arabic verbs and their subcategories.

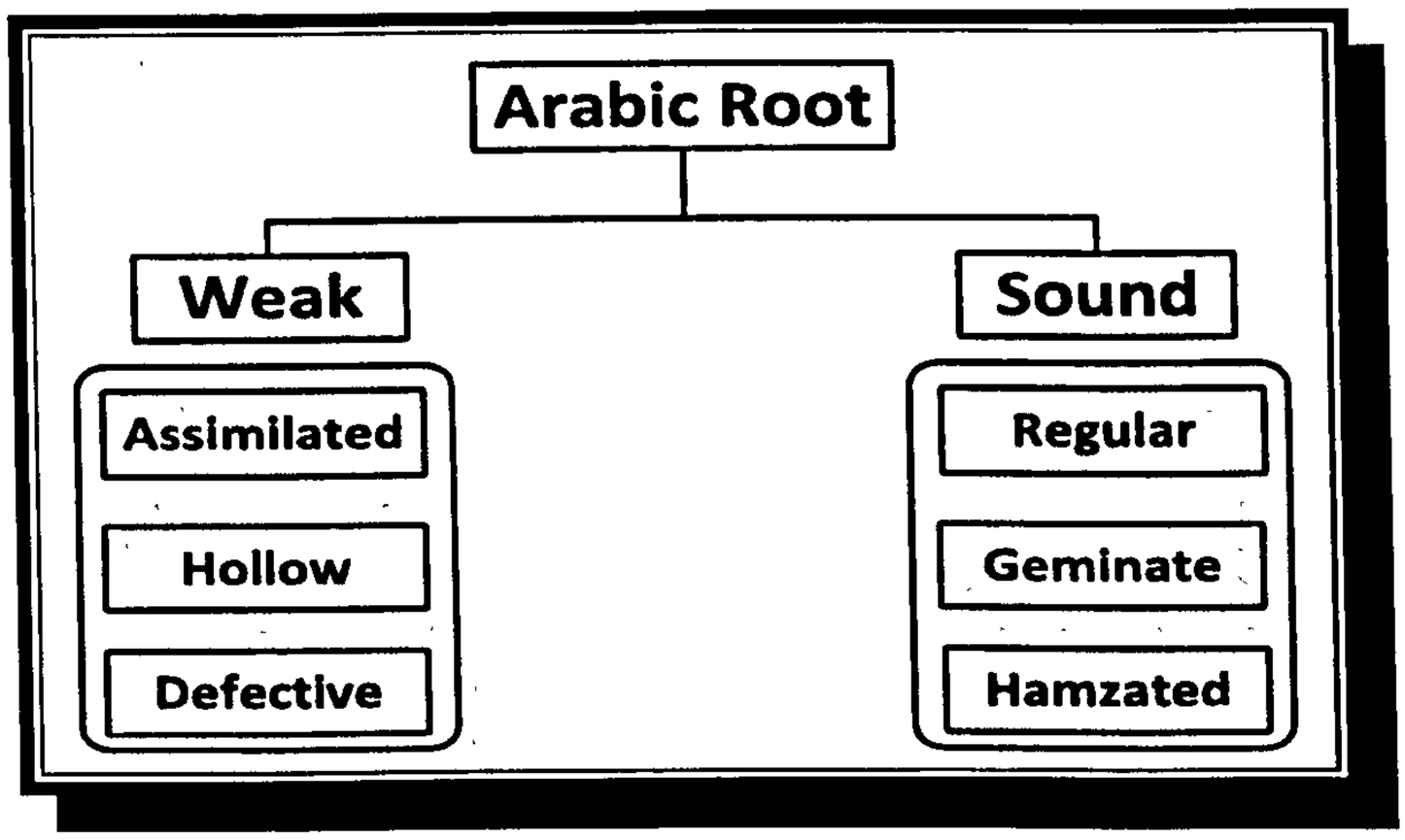


Figure 4.3: Classification of Arabic Verb Roots According to their Types

The sound verbs are further divided into three types: **regular, geminate,** and **hamzated** verbs. Firstly, the regular verbal roots consist of three consonants, all of which are different and none of them are vowels or hamza (glottal stop). For Example, [كتب](ktb,"write") and [درس](drs,"study") are sound root verbs. Secondly, the geminate doubled verbal roots are the ones where the second consonant is the same as the third one. In this case, the gemination mark is used to represent the combination of the identical letters into one stressed letter. For Example: [حلّ] (hll,"solve"). Finally, if the verb root has hamza [ء] in one of its consonants, then this type of root verbs is called hamzated verbs, such as [سأل](sĀl,"ask").

The weak root verbs are also divided into three types, according to the position of the vowel letters: **assimilated, hollow,** and **defective** verbal roots. Firstly, the assimilated verbs are the verbs that have a vowel as an initial letter, and the other letters are not vowels (consonants). For example: [وجد](wjd,"find") [وجد]. Secondly, the hollow verbs are the ones in which the

middle root letter is a vowel, such as, [باع](bA',"sell"). Finally, the defective verbs are those whose root has a vowel as a final letter, such as, [نسي] (nsy,"forget"). In many cases the Arabic word may contain one or more vowels, such as [طوي](Twy,"fold").

We analysed the triliteral root types in the four previous mentioned dictionaries as well as the **common dictionary**. The analysis results are summarised in Tables 4.8, 4.9, 4.10, 4.11 and 4.12 respectively. It is clearly observed from these tables that the overwhelming majority of Arabic root verbs are sound verbal roots, which are in average (**75%**) of all Arabic roots.

Root Type	No. of Roots	%
Regular	1,745	53.87%
Geminate	311	9.60%
Hamzated	272	8.40%
Sub Total	**2,328**	**71.87%**
Assimilated	192	5.93%
Hollow	449	13.86%
Defective	270	8.34%
Sub Total	**911**	**28.13%**
Total	**3,239**	**100.00%**

Table 4.8: Analysis of Mukhtar Al-Sahah Dictionary According to the Root Types

Root Type	No. of Roots	%
Regular	2,631	61.78%
Geminate	28	0.66%
Hamzated	326	7.65%
Sub Total	**2,985**	**70.09%**
Assimilated	269	6.32%
Hollow	594	13.95%
Defective	411	9.65%
Sub Total	**1274**	**29.91%**
Total	**4,259**	**100.00%**

Table 4.9: Analysis of Al-Ayen Dictionary According to the Root Types

Root Type	No. of Roots	%
Regular	3,475	63.16%
Geminate	434	7.89%
Hamzated	475	8.63%
Sub Total	**4,384**	**79.68%**
Assimilated	355	6.45%
Hollow	763	13.87%
Defective	0	0.00%
Sub Total	**1118**	**20.32%**
Total	**5,502**	**100.00%**

Table 4.10: Analysis of Taj Al-Aroos Dictionary According to the Root Types

Root Type	No. of Roots	%
Regular	3,763	57.36%
Geminate	486	7.41%
Hamzated	633	9.65%
Sub Total	**4,882**	**74.42%**
Assimilated	422	6.43%
Hollow	863	13.16%
Defective	393	5.99%
Sub Total	**1678**	**25.58%**
Total	**6,560**	**100.00%**

Table 4.11: Analysis of Lisan Al-Arab Dictionary According to the Root Types

Root Type	No. of Roots	%
Regular	3,763	57.36%
Geminate	486	7.41%
Hamzated	633	9.65%
Sub Total	**4,882**	**74.42%**
Assimilated	422	6.43%
Hollow	863	13.16%
Defective	393	5.99%
Sub Total	**1678**	**25.58%**
Total	**6,560**	**100.00%**

Table 4.12: Analysis of the Common Dictionary According to the Root Types

4.3 Arabic Patterns

Arabic root has a general basic meaning that forms the basis of many related meanings. These related meanings are represented by the root consonants put in different forms called **patterns**. Ryding [124] defines the morphological pattern as:

> **DEFINITION 4.2 (MORPHOLOGICAL PATTERN)**
>
> *"A pattern is a bound and in many cases, discontinuous morpheme consisting of one or more vowels and slots for root phonemes (radicals), which either alone or in combination with one to three derivational affixes, interlocks with a root to form a stem, and which generally has grammatical meaning." ([124], Page 48).*

As Definition 4.2 suggests, the pattern itself is nothing and has no meaning. It is just a template or a paradigm showing the positions of the root-letters where they should be. Therefore, based on this fact, the grammatical system of Arabic language is considered as a **root-and-pattern** based language.

The pattern usually is a combination of vowels, sometimes consonants, and "slots". Roots are interdigitated (interlocked) with the patterns in these slots to generate Arabic surface forms [55]. In literature, the commonly names for these surface forms are **stems**. So, the stem is the original form of the word before any linguistic transformation process is done [28].

For example, consider the Arabic root [كتب](ktb,"write"). This root presents the action of "writing". If this root interdigitates with the patten "$C_3C_2C_1$ه[3], then the Arabic word [مكتبة](mktbt) which means a library or a place to keep writings, is derived . Table 4.13 shows some examples of Arabic words derived from the root [كتب](ktb,"write") according to specific patterns. More details of word formation process are presented in Section 4.5 of this chapter.

[3]The symbol C_i is used here to present the slot for the root-letter, and the index i for the root-letter position.

Root	Pattern	Arabic Word	Translation
كتب	C_3اC_2C_1	كتاب	Book
ktb	$C_1C_2AC_3$	ktAb	
	C_3C_2اC_1	كاتب	Writer
	$C_1AC_2C_3$	kAtb	
	$C_3C_2C_1$ة	مكتبة	Library
	$mC_1C_2C_3t$	mktbt	
	$C_3C_2C_1$م	مكتب	Desk
	$mC_1C_2C_3$	mktb	
	C_3وC_2C_1م	مكتوب	Letter
	$mC_1C_2wC_3$	mktwb	
	$C_3C_2C_1$ي	يكتب	He is writing
	$yC_1C_2C_3$	yktb	
	C_3يC_2C_1	كتيب	Booklet
	$C_1C_2yC_3$	ktyb	

Table 4.13: Some Examples of Arabic Words Derived From the Same Root According to some Patterns.

Just as the roots are the base forms of the words, patterns also have their own base form. This form is defined by Arabic grammarians, and they have named it base-pattern-form[4] [124]. The base-pattern-form is referred to in Arabic as [وزن مجرد](wzn mjrd) meaning the morphologically simplest form or the stripped form [124]. The base-pattern-form for triliteral roots is the verb [فعل](f'l,"did"). The slots that were mentioned before, are replaced by the three letters of the base-pattern-form: [ف](f), [ع](‘), and [ل](l). From this base-pattern-form, all the other Arabic pattern forms are derived using the affix letters. The details of affix letters are presented in the next section.

The pattern is considered as the magic key in extracting the roots of Arabic words. If the pattern is identified for a given word, then it is easy to extract the root-letters for this word. For example, if we know that the corresponding pattern for the word [مكتبة](mktbt,"library") is [مفعلة](mf'lt) $mC_1C_2C_3t$, then the root-letters in the word are the letters that match the letters [ف](f), [ع](‘), and [ل](l) in the pattern. The matching letters in this example are [ك](k), [ت](t), and [ب](b). The combination of these letters produces the root [كتب](ktb,"write"). Figure 4.4 presents this process.

[4]Also is referred as Form I or Ground-Form.

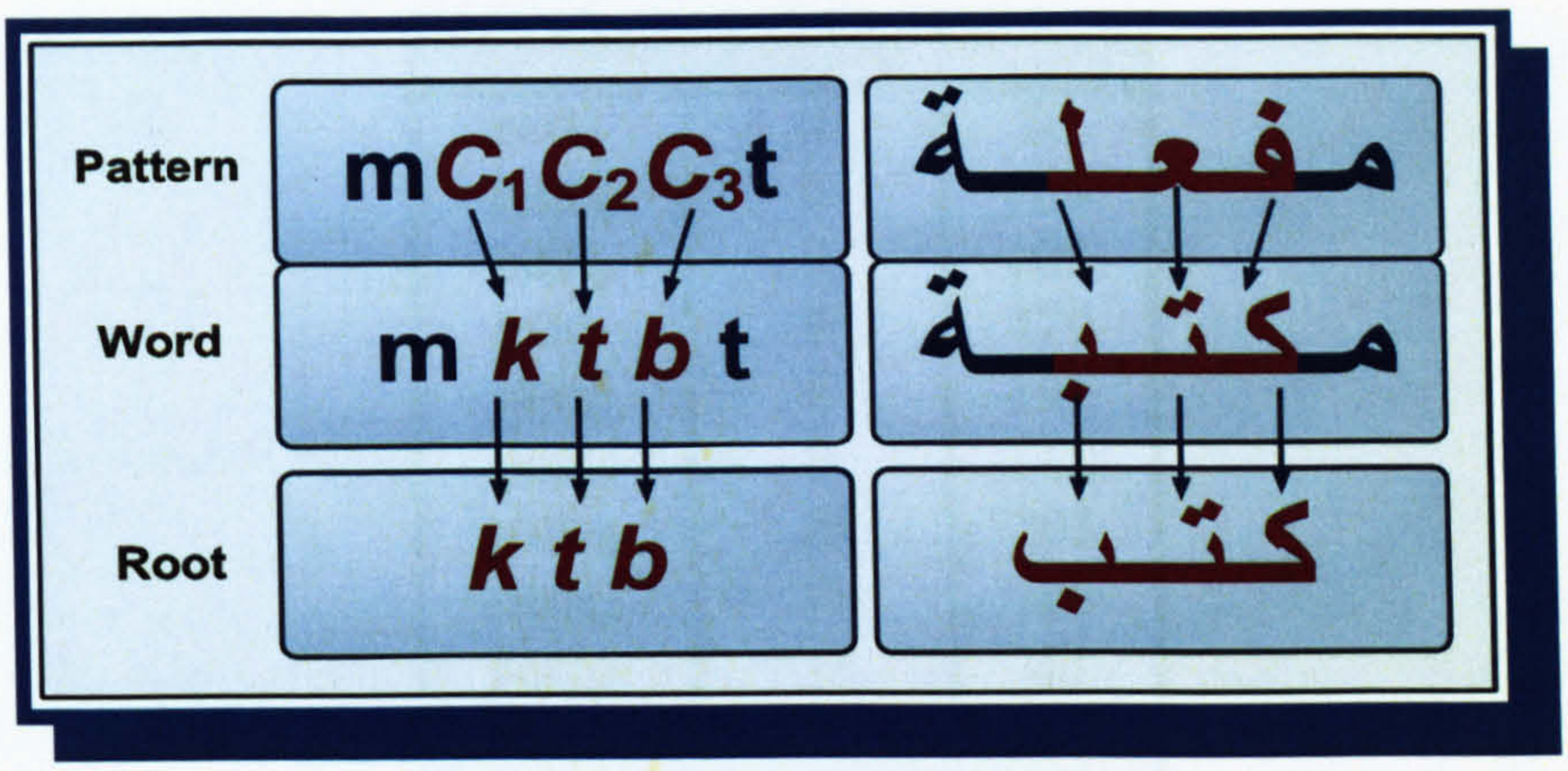


Figure 4.4: The Relation Between Pattern, Word, and Root

The main problem here is how to determine the correct pattern for the corresponding tested word. The conventional way, as it was mentioned in Chapter 3, is to compile a list of all allowable patterns in the language and then to check the tested word against all these patterns. This is time consuming, and in many cases more than one pattern is matched with the tested word.

The developed stemmer presented in this thesis is a computational approach, which avoids using a list of the affixes and patterns in the language. Instead of this, it tries to exploit the numerical relations between the letters that comprise the pattern. For this reason, we surveyed most of the conceivable derived-pattern-forms based on the triliteral roots. These surveyed patterns are compiled from Arabic grammar books [121][20][57][17][19]. These patterns calculated the total as **1,893** in number. To simplify the analysis of the patterns, the author classified the patterns according to their lengths. Each class is labeled as $patt_i$, where **i** refers to the pattern length. Table 4.14 demonstrates these classes and the number of patterns in each class. As a two examples, Table 4.15 and Table 4.16 list all the patterns of classes $patt_4$ and $patt_{12}$, respectively, while the full surveyed patterns are presented in Appendix A.

Pattern	Length	Total
$patt_4$	4	30
$patt_5$	5	136
$patt_6$	6	202
$patt_7$	7	179
$patt_8$	8	194
$patt_9$	9	712
$patt_{10}$	10	303
$patt_{11}$	11	105
$patt_{12}$	12	32
Total		1,893

Table 4.14: Classes of Arabic Patterns According to their Lengths

Pattern	Pattern Form	Pattern	Pattern Form
افعل	$AC_1C_2C_3$	بفعل	$bC_1C_2C_3$
تفعل	$tC_1C_2C_3$	تفعل	$tC_1C_2C_3$
فاعل	$C_1AC_2C_3$	فعال	$C_1C_2AC_3$
فعلا	$C_1C_2C_3A$	فعلت	$C_1C_2C_3t$
فعلة	$C_1C_2C_3t$	فعلة	$C_1C_2C_3t$
فعلة	$C_1C_2C_3t$	فعلة	$C_1C_2C_3t$
فعلك	$C_1C_2C_3k$	فعلن	$C_1C_2C_3n$
فعله	$C_1C_2C_3h$	فعلو	$C_1C_2C_3w$
فعلى	$C_1C_2C_3A$	فعلي	$C_1C_2C_3y$
فعول	$C_1C_2wC_3$	فعيل	$C_1C_2yC_3$
ففعل	$fC_1C_2C_3$	فوعل	$C_1wC_2C_3$
فيعل	$C_1yC_2C_3$	كفعل	$kC_1C_2C_3$
لفعل	$lC_1C_2C_3$	مفعل	$mC_1C_2C_3$
نفعل	$nC_1C_2C_3$	وفعل	$wC_1C_2C_3$
يفعل	$yC_1C_2C_3$	أفعل	$AC_1C_2C_3$

Table 4.15: List of Arabic Patterns $patt_4$

As mentioned earlier, the root is usually consist of three letters, and the pattern indicates the ordering and the positions of these letters in addition to the positions of the non root-letters. In order to investigate the positions of the root-letters, the author studied all the compiled patterns and computed what is so-called the **root-distance**. The **root-distance** is defined as the measure of counting the number of letters between any two root-letters and

Pattern	Pattern Form	Pattern	Pattern Form
الاستفعاليون	$AlAstC_1C_2AC_3ywn$	باستفعالاتها	$bAstC_1C_2AC_3AthA$
بالافتعاليات	$bAlAC_1tC_2AC_3yAt$	بالمستفعليات	$bAlmstC_1C_2C_3yAt$
بالمفعولاتية	$bAlmC_1C_2wC_3Atyt$	فاستفعلتماها	$fAstC_1C_2C_3tmAhA$
فاستفعلتموها	$fAstC_1C_2C_3tmwhA$	فاستفعلتموهم	$fAstC_1C_2C_3tmwhm$
فالاستفعالات	$fAlAstC_1C_2AC_3At$	فالافتعاليات	$fAlAC_1tC_2AC_3yAt$
فبالمستفعلات	$fbAlmstC_1C_2C_3At$	فبالمفعوليات	$fbAlmC_1C_2wC_3yAt$
كالمفعولاتية	$kAlmC_1C_2wC_3Atyt$	لاستفعالاتنا	$lAstC_1C_2AC_3AtnA$
لاستفعالاتنا	$lAstC_1C_2AC_3AtnA$	لاستفعالاتها	$lAstC_1C_2AC_3AthA$
واستفعالاتكم	$wAstC_1C_2AC_3Atkm$	واستفعالاتهم	$wAstC_1C_2AC_3Athm$
واستفعلتماها	$wAstC_1C_2C_3tmAhA$	واستفعلتموها	$wAstC_1C_2C_3tmwhA$
واستفعلتموهم	$wAstC_1C_2C_3tmwhm$	وافتعالياتها	$wAC_1tC_2AC_3yAthA$
وافتعالياتها	$wAC_1tC_2AC_3yAthA$	والاستفعالات	$wAlAstC_1C_2AC_3At$
والاستفعالية	$wAlAstC_1C_2AC_3yt$	والافتعاليات	$wAlAC_1tC_2AC_3yAt$
والافتعاليون	$wAlAC_1tC_2AC_3ywn$	والافتعاليين	$wAlAC_1tC_2AC_3yyn$
والفعالينيات	$wAlC_1C_2AC_3ynyAt$	واللامفعولية	$wAllAmC_1C_2wC_3yt$
والمفعولاتية	$wAlmC_1C_2wC_3Atyt$	وبالمفعوليات	$wbAlmC_1C_2wC_3yAt$

Table 4.16: List of Arabic Patterns $patt_{12}$

is computed as shown in Equation 4.1.

$$root\text{-}distance(i,j) = j - i - 1 \tag{4.1}$$

where **i** and **j** are the positions of the two root-letters and $j > i$. For example, the pattern $mC_1AC_2C_3$ has five letters, three of them are root-letters ($C_1C_2C_3$) which occupies the positions 2, 4, and 5 respectively, and the other two are non root-letters (**m, A**) which occupies the positions 1 and 3. The root-distance between C_1 and C_2 is **1** = (4 - 2 -1), between C_2 and C_3 is **0** = (5 - 4 -1), while between C_1 and C_3 is **2** = (5 - 2 -1).

The computation of the root-distance is useful in some cases. In the case of two root-letters being extracted and their orders in the root are not known, the root-distance value can assist of extracting the third one. For example, if the value of the root-distance is two or more, then this means that the current two root-letters present the first and the last ones (C_1C_3). As a consequence, the other root-letter (C_2) should be between the identified two letters. This analysis is based on the fact that only one or two letters can be added inside the root-letters [133]. Table 4.17 shows the maximum value of the root-distance in each pattern class and the number of letters that are found inside the root-

letters.

Pattern Class	Max. Value of Root-Distance	No. of Letters
$patt_4$	2	1
$patt_5$	3	2
$patt_6$	4	3
$patt_7$	4	3
$patt_8$	4	3
$patt_9$	4	3
$patt_{10}$	4	3
$patt_{11}$	3	2
$patt_{12}$	3	2

Table 4.17: Summary of the Values of the Root-Distances in each Pattern Class

An interesting fact found during the Arabic pattern analysis is that the number of letters inside the root-letters in some Arabic words may be three letters. This is clear in Table 4.17 in pattern classes $patt_6$, $patt_7$, $patt_8$, $patt_9$, and $patt_{10}$. We found that all of these words are derived from the Arabic pattern [فواعيل] ($C_1wAC_2yC_3$). The analysis of this pattern clearly shows that these three letters are: [ا](A), [و](w), and [ي](y) which they are the vowels in Arabic language.

The Arabic pattern analysis which is presented in this section is used as the core of the **Phase I** in the developed stemming algorithm. The complete details of the **Phase I** are presented in Chapter 5.

4.4 Arabic Affixes Analysis

4.4.1 Linguistic Analysis

Most Arabic words contain some kind of affixation. Affixation is a process whereby one or several affixes are added to a root to create either an inflected word form or a derived word form. In this work, the term **affixes** is used

to represent any non root-letters, either they are derivational or inflectional letters. The inflectional letters are the marks of tense (past, present or future), gender (masculine, feminine or no gender), and/or number (singular, dual, plural). They also include some prepositions, conjunctions, determiners, pronouns, and possessive pronouns. In some cases all of these affixes can be found in one word as in the word [والمدافعون] (wAlmdAf'wn, "and the defenders"). This word has ten letters, three of them are root-letters, while the others are affixes. The root of this word is [دفع](df',"defend"). The decomposition of this word to its morphemes (root and affixes) is shown in Figure 4.5.

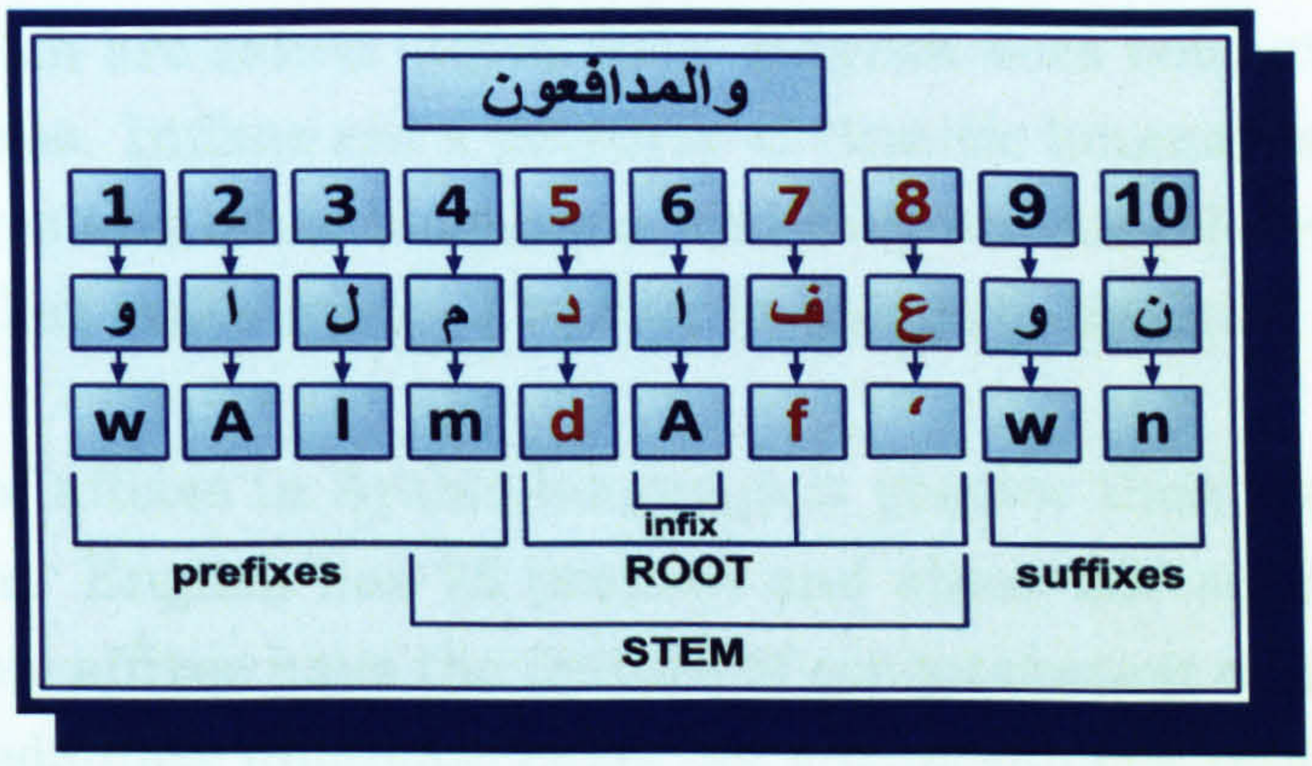


Figure 4.5: Example of Word Decomposition

The example in Figure 4.5 can clearly deduce the differences between the three main terms used in computational linguistics : roots, stems and affixes. As mentioned in Section 4.2 the roots have more concrete meanings, while affixed words are more abstract. A root is the base form of a word which can not be further analysed without the loss of the word's identity. Or it is that part of the word left when all the affixes are removed. On the other hand, stem refers to the portion of an affixed form with the affix which can be removed. From a purely morphological point of view, the building up of stems is accomplished through the addition of one or more affixes to the root using a pattern.

To illustrate the above, Figure 4.5 is suffice for the present. This Figure shows that the word [والمدافعون](wAlmdAf'wn) is composed of the stem [مدافع]

(mdAf', "defender") preceded by the word-prefixes the conjunction [و](w,"and"), and the definite article [ال](Al,"the"). This word is also succeeded by the suffix pronoun [ون] (wn,"them"[masculine plural]) while, the stem [مدافع] (mdAf') in turn is composed of the [م](m) and the infix [ا](A) plus the root [دفع](df',"defend").

Depending on the position of the affixes in relation to the root the affixes in Arabic language are classified as prefixes, suffixes, and infixes. Arabic prefixes are the sets of letters attached to the beginning of the roots and written as part of them. Suffixes are the sets of letters attached to the end of the root written as part of them [26]. In addition, some affixes are inserted inside the root script which are called infixes [61]. English does not have any good examples of infixes. Infixes are a property of Semitic languages that are differ these languages and other languages, and they are considered as "one of the complexities that make Arabic a harder language to analyse" [133].

The number of affixes in Arabic language is greater than those used in English language. English has **75** prefixes and about **250** suffixes [125] (cited by [28]). Arabic affixes have the feature of concatenating with each other according to predefined linguistic rules, as a consequence this is increase the overall number of affixes [28]. The author successfully compiled **204** Arabic prefixes, **7** infixes and **193** suffixes. The maximum length of the prefix is found seven letters, while for the suffix it is six letters. The extraction process of these affixes is simply done by analysing the pattern classes ($patt_4$ to $patt_{12}$), based on dividing the pattern letters into two types: root-letters and non root-letters (affixes). The pattern classes have been discussed in Section 4.3 of this chapter.

In summary, an affix is a morpheme that can be added to a root or a stem to form new words or meanings. It is important to clear the role of affixes in Arabic language compared with other languages. In English language the affix plays the role of a modifier of the meaning. For example, if the English word **happy** prefixed with **un**, a new word with a new meaning will be generated (**unhappy**). In contrast, the affix in Arabic language in addition plays the same role as in English language, it can add an entity to the word that is

attached with. For example, if the Arabic word [مسح](msH,"clean") is suffixed with [ه](h,"pronoun"), then the new generated word is [مسحه](msHh) which means "he cleaned it".

4.4.2 Affixes Extraction

The affixes extraction process starts by encoding each Arabic pattern using a binary code. The root-letters in the patter are encoded by **zero**, while the other letters (affix letters) are encoded by **one**. For example, the pattern[يفاعلون] (yfA'lwn) which belongs to class $patt_7$ are encoded as **1010011**. Since, the positions of the root-letters are: 2, 4, and 5. Figure 4.6 details the encoded procedure for this pattern.

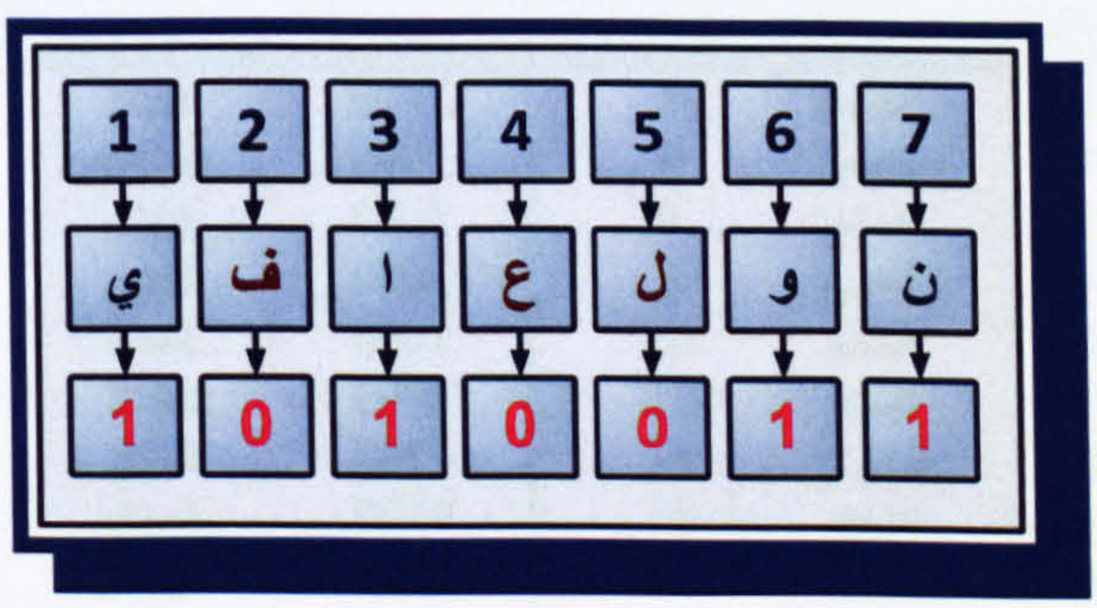


Figure 4.6: Encoded Example for the Arabic Pattern [يفاعلون](*yfA 'lwn*)

It is clear from Figure 4.6, the prefix of this pattern is [ي](y), the infix is [ا](A), and the suffix is [ون](wn). We follow the same previous steps to extract all the affix letters in all compiled patterns and divide them to prefixes, infixes, and suffixes. The pseudo code for this process is listed in Algorithm 4.1.

ALGORITHM 4.1 : Pseudo Code for Affixes Compilation Process

```
1: Initialise NZeros.
2: for all pattern class patt_i do
3:     for all pattern belongs to patt_i do
4:         Encode pattern followed the previously encoded procedure.
5:     end for
6:     - Extract the pattern letters that have 1's in the encoded string and
7:       precede the first 0; add these letters to the Prefixes List.
8:
```

9: - Extract the **pattern** letters that have **1's** in the encoded string and
10: succeed the last **0**; add these letters to the **Suffixes List**.
11:
12: - Extract the **pattern** letters that have **1's** in the encoded string,
13: which succeed the first **0** and precede the last **0**; add these letters to the **Infixes List**.
14: **end for**

The way how the algorithm works is explained by using an example to compile the affix letters for the pattern class $patt_4$. There were **30** patterns in this class (see Table 4.15). Each pattern is encoded and followed the same encoded procedure which was presented in the previous page. Table 4.18 shows all the patterns in $patt_4$ and their encoded strings.

No.	Pattern Form	Pattern Code	No.	Pattern Form	Pattern Code
1	فعلا	0001	2	فعلت	0001
3	فعلك	0001	4	فعلن	0001
5	فعله	0001	6	فعلو	0001
7	فعلى	0001	8	فعلي	0001
9	فعلة	0001	10	فعلة	0001
11	فعلة	0001	12	فعلة	0001
13	فعال	0010	14	فعول	0010
15	فعيل	0010	16	فاعل	0100
17	فوعل	0100	18	فيعل	0100
19	افعل	1000	20	بفعل	1000
21	تفعل	1000	22	ففعل	1000
23	كفعل	1000	24	لفعل	1000
25	مفعل	1000	26	نفعل	1000
27	وفعل	1000	28	يفعل	1000
29	تفعل	1000	30	أفعل	1000

Table 4.18: List of Arabic Patterns $patt_4$ and their Encoded Strings

Now, to extract the list of prefixes in $patt_4$, all the patterns that have a character **one** and precede the first **zero** are retrieved (Step 6 of Algorithm 4.1). This follows the definition of prefix letters. The patterns that have met this condition in Table 4.18 are: 19, 20, ..., 30. For each retrieved pattern, the sub-letters that precede the first **zero** are extracted and added to the prefix letters list of the class $patt_4$. Note that these prefixes occupy the first position for all the retrieved patterns. The list of prefix letters in $patt_4$ are: {[ا](A),

[ب](b), [ت](t), [ي](y), [ف](f), [ك](k), [ل](l), [م](m), [ن](n), [و](w), [أ](Ā)}.

To extract the list of suffixes in $patt_4$, all patterns that have the character **one** and succeed the last **zero** are retrieved (Step 9 of Algorithm 4.1). This is followed by the definition of suffix letters. The patterns that met this condition in Table 4.18 are: 1, 2, . . . , 12. For each retrieved pattern, the sub-letters that succeed the last **zero** are extracted and added to the suffix letters list of the class $patt_4$. Note that the positions of these prefixes are in the last part for all the retrieved patterns. The list of suffix letters in $patt_4$ are: {[ا](A), [ت](t), [ك](k), [ن](n), [و](w), [ي](y), [ة](t)}.

Following the same procedure, we can easily extract the list of infix letters from Table 4.18 for $patt_4$. These letters are in patters: 13, 14, . . . , 18. The positions of infixes are in the middle between the first zero and the last zero. The list of infix letters in $patt_4$ are: {[ا](A), [و](w), [ي](y)}. These are the long vowels of Arabic language. Table 4.19 shows the affixes of $patt_4$ by combining the previous list of prefixes, infixes, and suffixes.

Prefixes			Infixes			Suffixes		
ا	ب	ت	ا	و	ي	ا	ت	ك
ي	ف	ك				ن	هـ	و
ل	م	ن				ي	ة	
و	أ							

Table 4.19: The Affixes of $patt_4$

We followed the same steps of Algorithm 4.1 in order to extract all the affix letters of Arabic for all the remaining pattern classes. Table B.8 shows the affixes of the category $patt_{12}$ while Section 1 of Appendix B shows the complete lists of the extracted affixes for all categories.

As mentioned earlier, the main function of the morphology process is to decompose the Arabic word into its morphological components: prefixes, stem, and suffixes. The combination of any of these prefixes and suffixes is not nec-

Prefixes			Infixes			Suffixes		
الاست	باست	بالا	ا	و	ي	ات	اتكم	اتنا
بالم	بالمست	فاست	ت			اتها	اتهم	اتية
فالا	فالاست	فبالم				تماها	تموها	تموهم
فبالمست	كالم	لاست				يات	ياتها	ية
وا	واست	وال				ينيات	يون	يين
والا	والاست	واللام						
والم	وبالم							

Table 4.20: The Affixes of $patt_{12}$

essarily valid or legal. For example, with returning back to the Table 4.16, it is clear the prefix [الاست] (AlAst) could be combined with the suffix [يون](ywn) in the same word, and this combination is considered as a valid combination. The combinations between this prefix and the other suffixes lead to an invalid combinations. Another example, the prefix [فاست] (fAst) in the same table make a valid combination if it is combined with one of the following suffixes: [تماها] (tmAhA), [تموها](tmwhA), or [تموهم](tmwhm).

In order to confirm the valid combination of the prefixes and the suffix, all the the compiled affixes (see Section 1 of Appendix B) are analysed with respect to the compiled Arabic patterns (see Appendix A). The analysis unfolds the valid prefix-suffix combinations. This validation is used in Chapter 7 to check the correctness of the extracted root and the affix letters attached to it. The valid prefix-suffix compositions of all categories are presented in Section 2 of Appendix B, while Table 4.21 shows an example of the prefix-suffix combination list of the the category $patt_{12}$.

4.4.3 The Frequency of Arabic Affix Letters

Arabic affix letters have been identified by Arabic linguists. The commonly used affix letters [121] are presented in Table 4.22 and they are rounded up in the word [سألتمونيها](sĀltmwnyhA "you asked me for/about it"). In some cases the definite article [ال](Al) may be prefixed by one of the following letters, {[ك](k), [ف](f), [ب](b))} to compose [كال](kAl), [فال](fAl), and [بال](bAl). In this case, these letters become as a part of the prefixes when they are attached

Prefix	Suffixes				
الاست	يون				
باست	اتها				
بالا	يات				
بالم	اتية				
بالمست	يات				
فاست	تماها	تموها	تموهم		
فالا	يات				
فالاست	ات				
فبالم	يات				
فبالمست	ات				
كالم	اتية				
لاست	اتنا	اتها			
وا	ياتها				
واست	اتكم	اتهم	تماها	تموها	تموهم
وال	ينيات				
والا	يات	يون	يين		
والاست	ات	ية			
واللام	ية				
والم	اتية				
وبالم	يات				

Table 4.21: The Prefix-Suffix Combinations of $patt_{12}$

at the beginning of the Arabic words. For the purpose of this study, the letters {[ك](k), [ف](f), [ب](b))} are added to the affix of letter set, and the final set of affix letters is presented in Table 4.23.

س	أ	ل	ت	و	م	ن	ي	هـ	ا
s	Ā	l	t	w	m	n	y	h	A

Table 4.22: The Original Set of Affix Letters

س	أ	ل	ت	و	م	ن	ي	هـ	ا	ك	ف	ب
s	Ā	l	t	w	m	n	y	h	A	k	f	b

Table 4.23: The Modified Set of Affix Letters

A program was implemented for the purpose of analysing the frequency of the affix letters in Arabic language. An annotated corpus[5] containing **66,344**

[5]The annotated corpus is discussed in Chapter 6.

Arabic words was used in this statistical analysis. It was found that the most frequent affix letter is [ا](**A**) which appears **66,862** times in different positions of the Arabic words, then the letter [ي](**y**) which occurs **24,694** times, and the lowest frequently used letters are [ف](**f**), [ب](**b**) and [ك](**k**) which appear about **1,345**, **1,212** and **1,193** times respectively. The analysis of the frequency distribution of these affixes is shown in Table 4.24. This table also contains in addition the previous affixes, some other letters which they represent different shapes of some of the affix letters such as {[أ](Ā), [إ](Ą), [آ](), [ى](Ã), [ء](’), [ؤ](w̄), [ة](t)}.

The above analysis and facts will be the backbone of the developed computational approach to extract the roots of Arabic words. This will be discussed at length in Chapters 5 and 6. Also, such facts and statistics are very useful in different computational and linguistic studies.

Letter	Transliteration	Frequency
ا	A	66,862
ي	y	24,694
و	w	23,167
ت	t	21,025
ن	n	19,170
م	m	12,156
ل	l	9,688
ه	h	9,501
ة	T	9,320
إ	b	4,558
س	s	1,941
أ	k	1,704
ى	y	1,623
ئ	Ã	1,527
ء	’	1,467
ؤ	w̄	1,403
آ	A	1,351
ف	f	1,345
ب	b	1,212
ك	k	1,193

Table 4.24: Arabic Affix Letters Frequencies

4.5 Word Formation

The previous sections discussed the Arabic morphology terms: root, stem, pattern, and affix separately. This section combines these terms and presents their roles in Arabic word formation process.

4.5.1 The Definition of the Arabic Word

The literature in Arabic language does not provide a standard definition of the meaning of the Arabic word term. Some researchers defined it orthographically as a unit that is surrounded by two white spaces. The most common definition of the Arabic word is the one that was presented by Al-Hamlawee [17].

DEFINITION 4.3 (ARABIC WORD)

An Arabic word is the single and isolated lexeme that represents a certain meaning abstract written sequence. [17]

An Arabic word might be an original Arabic word, or a borrowed word [12]. The **original Arabic words** are divided into two sub-categories:

1. DERIVATIVE ARABIC WORDS These are the words that are derived from roots according to derivation rules in Arabic. The majority of these words are nouns and verbs.

2. FIXED ARABIC WORDS These are the words which do not comply with the derivation rules in Arabic. The majority of these words are pronouns, prepositions, and conjunctions. Fixed Arabic

words are a minority when compared to derived words.

Borrowed words, also referred as **Arabised words**, are words that are adopted from other languages like English language [111]. These words are adapted to conform with the Arabic word paradigms, and become part of the Arabic lexicon [111]. For example, the borrowed word [كمبيوتر] (kmbywtr), which is commonly used in Arabic texts these days, is adapted from the English word [Computer]. Borrowed words have no roots, and are not derived by any derivation rule in Arabic.

4.5.2 Word Formation Process

As mentioned earlier, Arabic is a root-pattern type of language. Verbs, nouns, and adjectives in Arabic languages are derived from a root by using different patterns and affixes. The root is usually the only common element shared by derivationally related forms.

A stem is generated from a bare root when it is combined with a given pattern [27]. Each pattern in Arabic language produces a stem. Stems derived from the same root share an aspect of meaning. The addition of prefixes, and suffixes to the stem produces the Arabic word. So, the stem is generated through the interdigitation of the root with a pattern having zero or more affixes. The stem itself is considered as a valid Arabic word.

The **base-pattern-form** of Arabic language is the three-letters verb [فعل] (f'l,"did"). The interdigitation process is done by mapping the first consonant of the root to the letter [ف](f), the second consonant of the root to the letter [ع](’), and the third consonant of the root to the letter [ل](l). The order of these three consonants does not change.

As mentioned in Chapter 1, Arabic morphology is a highly structured process [133]. For example, the pattern $mC_1C_2C_3t$ is a paradigm that is used to generate words refer to the place where the activity specified by the root verb

occurs. So, if the Arabic root [درس] (drs, "study") is interdigiated with this pattern, then the Arabic word [مدرسة] (mdrst, "school") is generated indicates the place of the studying. Now, if we want to generate a new word indicates "the place of farming", then just the interdigiation of the root [زرع] (zr',"plant") with the same pattern should be applied. And, to generate a word indicates "the place of writing", then the interdigiation of the root [كتب] (ktb,"write") with the same pattern should be applied. A schematic diagram for this procedure is shown in Figure 4.7.

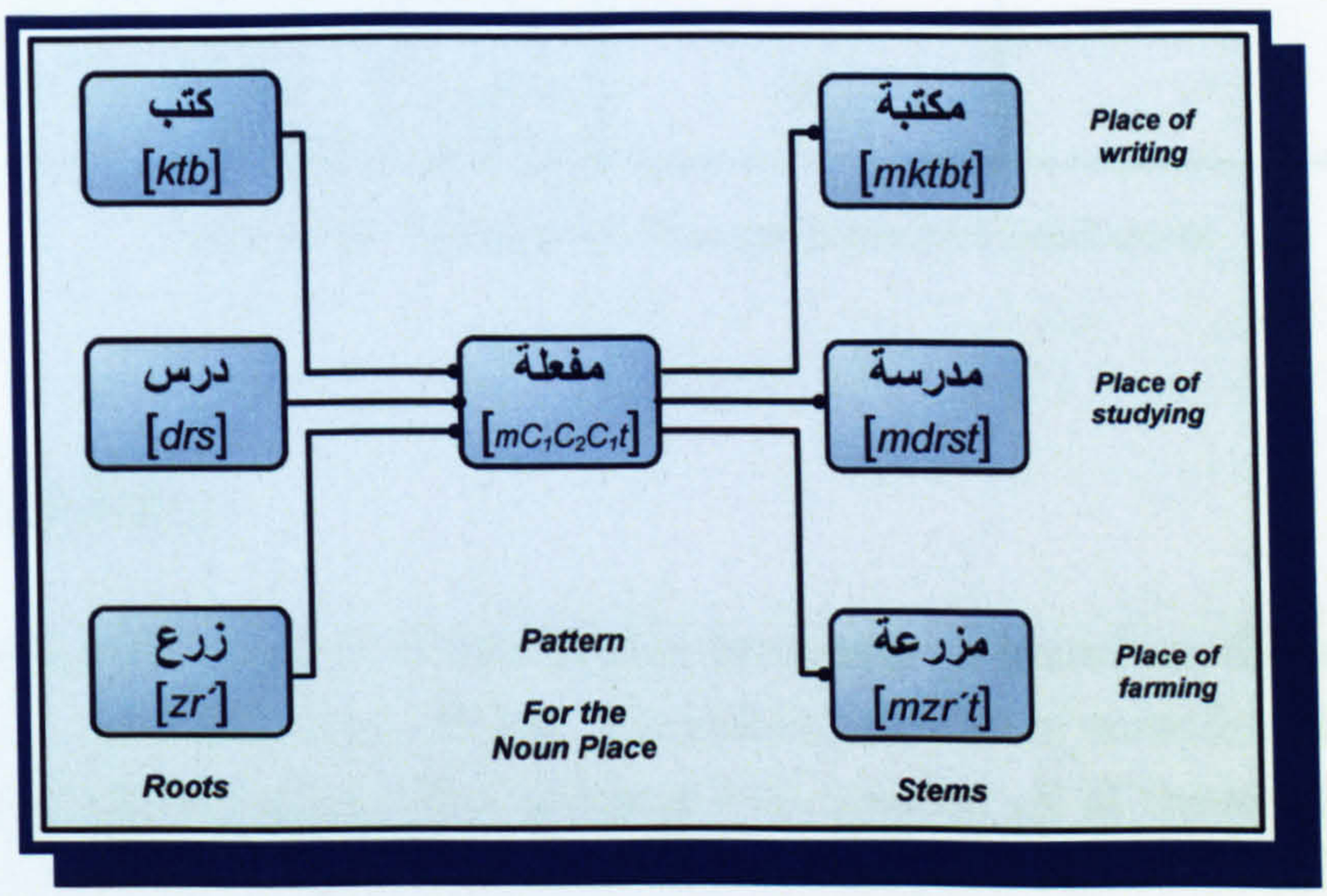


Figure 4.7: Formation Process of Arabic Stems and Words

Another example that presents the structured of Arabic morphology is illustrated here. Suppose we want to generate an Arabic word that is specified that two people are playing to each other, the root [لعب] (l'b,"play") is used. The suffix [ان](An) is added to it to indicate that the verb is being done by two people; the prefix [ي](y) is added to indicate that the verb is in the present tense; and the infix [ا](A) is added to indicate that they are playing to each other. Thus the verb obtained is [يلعبان] (yl'bAn). If the same meaning is to be conveyed but, now, instead of the two people are playing to each other we want to say that they write with each other, it is only necessary to replace the letters of the verb root play [لعب](l'b) with those of the verb root write [كتب](ktb) to obtain the verb [يكتبان](yktbAn).

Unfortunately, the formation of Arabic words is not usually a straightforward process. In Arabic morphological derivations, many cases of the root-letters are modified or deleted during the formation process. This is done specially in the words that are generated from weak roots. Some examples of this are shown in Table 4.25.

Root	Pattern	Word	Deleted Letter (from the root)	Changed Letter (from the root)
[أمن] $\bar{A}mn$	[يفعلون] $yC_1C_2C_3wn$	[يؤمنون] $y\bar{w}mnwn$	-	[أ] →[ؤ] $\bar{A}$ →$\bar{w}$
[هدي] hdy	[يفعلون] $yC_1C_2C_3wn$	[يهتدون] $yhtdwn$	[ي] y	-

Table 4.25: Example Of Changed/Deleted Root-Letters

4.6 Summary

The grammatical system of the Arabic language is based on three main concepts, roots, pattern, and affixes. Therefore, Arabic is considered as a **root-pattern** based language. This chapter has covered all of these concepts, and has discussed their roles in the word formation process.

The roots in Arabic language contain the important part of the meaning of the words. The usual number of consonants in the Arabic root is three (**triliteral**). However, there are also biliteral, quadriliteral, and pentaliteral roots. The triliteral roots are the overwhelming majority of Arabic roots, and to a lesser extent, quadrilateral, pentaliteral, and hexaliteral. These results were conducted through a comprehensive analysis of four famous Arabic dictionaries. These dictionaries were compiled by hand by the author. In consequence, a common dictionary was created from these four dictionaries. The common dictionary provides a unique set of roots for other researches in the field and is therefore a significant contribution from this work.

The Arabic patterns are morphological templates showing the positions of the

root-letters where they should be. This fact has motivated us to exploit the numerical relations between the letters that comprise the pattern. We have successfully surveyed **1,893** conceivable pattern forms based on the triliteral roots. The surveyed patterns have been anlaysed and classified into nine classes according to their lengths. Also, the relations between the root-letters were studied based on the grammatical Arabic patterns.

The affixes are morphemes that can be added to a root to form new words or meanings. Depending on the position of the affixes in relation to the root, the affixes in Arabic language are classified into, prefixes, suffixes, infixes. Arabic prefixes are the sets of letters attached to the beginning of the root. Suffixes are the sets of letters attached to the end of the root. In addition, some affixes are inserted inside the root script which are called infixes. In this chapter, the author compiled **204** prefixes, **7** infixes, and **193** suffixes. Also, the analyses of these affixes was presented in this chapter. Furtherer more, the valid combinations of the prefix-suffix were discussed.

The Arabic words might be original words, or borrowed words. The original words are those that are derived from roots according to derivation rules in Arabic while, the borrowed words are those words that are adopted from other languages for example English language and have become part of the Arabic lexicon.

The last part of this chapter discussed the Arabic word formation process. This process usually is a straightforward process and is done through the interdigitation of the root-letters with a vocalic pattern in the root slots which then might be accompanied by the affixes set. An Arabic word could have repeated forms and combinations of these affixes, i.e., more than one affix may be attached to a word.

The next chapter discusses the **Phase I** of the computational approach of the extraction of the Arabic word roots.

Chapter 5

COMPUTATIONAL APPROACH TO ARABIC ROOT EXTRACTION - PHASE I

5.1 Introduction

This chapter presents **Phase I** of the new MUAIDI-STEMMER approach for extracting Arabic word roots. This approach is a computational model attempting to exploit the numerical relations between Arabic letters, and avoiding having a list of the root and pattern of each word in the language. Furthermore, this approach does not use any set of linguistic rules.

The MUAIDI-STEMMER approach consists mainly of two phases: (a) **Phase I**, and (b) **Phase II**. To extract the root for a given word, one phase is triggered depending on the number of root-letters in the word. **Phase I** which is presented in this chapter, is based on the analysis discussed in the previous chapter, while **Phase II** which is presented in Chapter 6, is based on an artificial neural network trained by backpropagation learning rule.

5.2 Overall Structure of MUAIDI Stemmer

The schematic architecture of the MUAIDI-STEMMER system for the extraction of the Arabic word roots is illustrated in Figure 5.1. The architecture model mainly consists of two main and four supplementary modules. The main modules are **Phase I** and **Phase II**. The supplementary modules are utility modules that put the text in a form suitable for stemming. The supplementary modules are interconnected, i.e., outputs of one module are the inputs of the other. These modules, which are listed below, are explained in the order in which they are performed in the following subsections:

- Cleansing Module
- Word Tokeniser Module
- Stopwords Removal Module
- Normalisation Module
- Weights Module

5.2.1 Cleansing Module

The main function of this module is to leave the Arabic scripts and remove everything else such as, punctuation symbols, numeric data, dates, and times. Furthermore, this module will detect and remove all the Arabic diacritics founded in the text (except the gemination mark). As was mentioned in Chapter 2, the gemination mark is a symbol that is placed above the Arabic letter to indicate a double consonant while pronouncing it. The cleansing module removes this mark and doubles the letter that is bearing it. For example, [علّم] → [عللم].

The resultant cleaned data will then be fed to the next module. The need for data cleaning is center around improving the quality of data to make them "fit for use". Figure 5.2 shows an example of Arabic text before cleansing, while

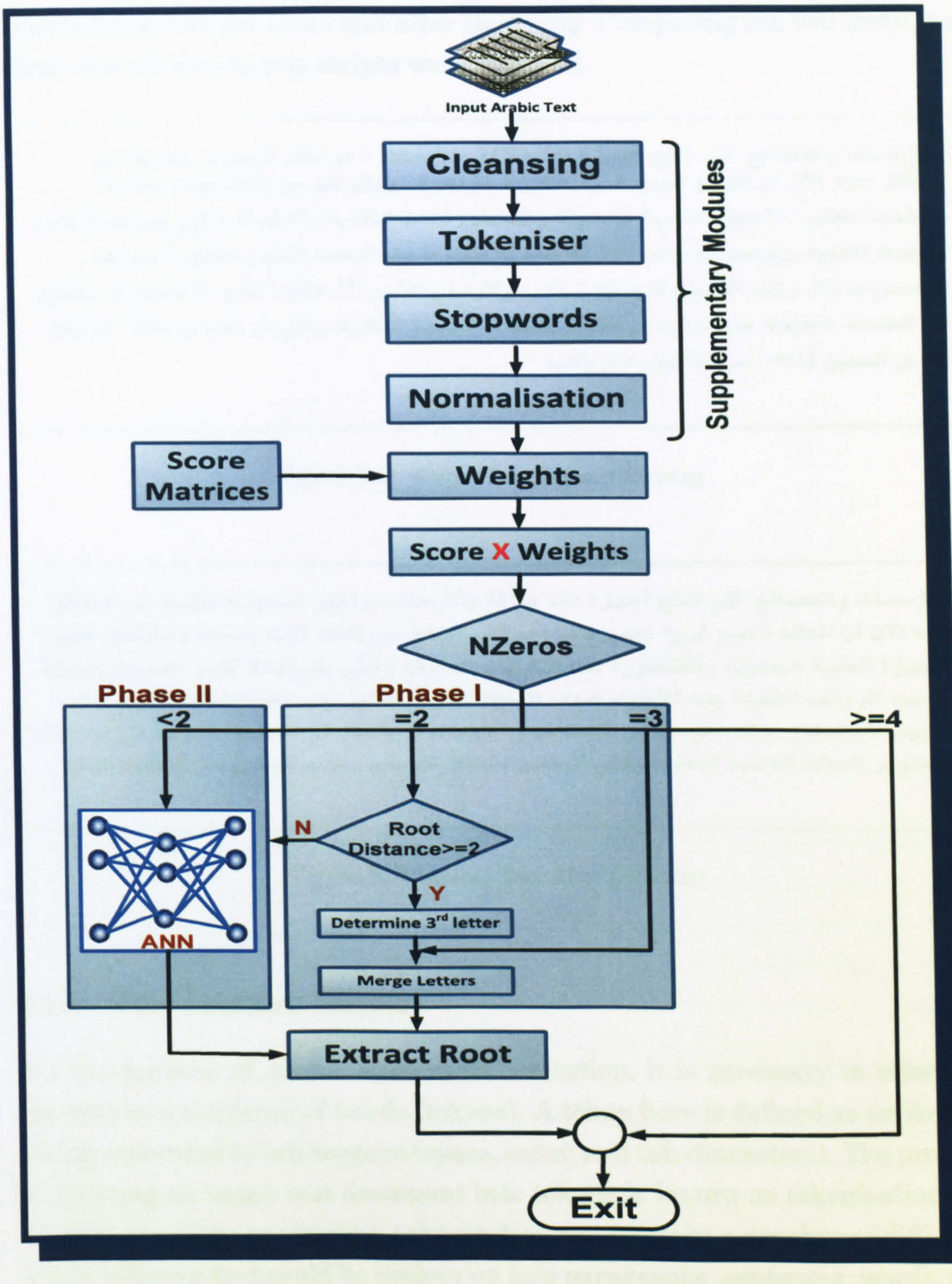


Figure 5.1: The Overall Architecture of the Developed Computational Model

Figure 5.3 shows the same text after cleansing. Comparing the two texts, it is clear that all non-Arabic scripts were removed.

المؤسسات والمصانع التي تنتج أجهزة ذات علاقة بالأبحاث. * مراكز البحوث الطبية من المتفق عليه الآن أن العالم أصبح قرية صغيرة مزدهرة ، وبالرغم من ذلك فإننا بحاجة وتنقصنا كثيراً الأجهزة الطبية الحديثة والمتطورة ذات التقنيات الدقيقة والتي بدونها لا تثمر البحوث الطبية، وبسبب الرواتب القليلة جداً للأطباء ضعف البحث العلمي كثيراً، وبرغم التحسن ما زال راتب الطبيب لا يتجاوز ٥٠١ دولاراً في الشهر!!!، فكيف تنمو الأبحاث، أو يندفع في الخدمة، ناهيك من البحوث وتطوير الحالة العلمية المادية. يكفي أن نضرب مثلاً: خريجو إحدى السنين (١٩٩١) من الأطباء ٤٦٨ طالباً

Figure 5.2: Arabic Text Before Cleansing

المؤسسات والمصانع التي تنتج أجهزة ذات علاقة بالأبحاث مراكز البحوث الطبية من المتفق عليه الان أن العالم أصبح قرية صغيرة مزدهرة وبالرغم من ذلك فإننا بحاجة وتنقصنا كثيرا الأجهزة الطبية الحديثة والمتطورة ذات التقنيات الدقيقة والتي بدونها لا تثمر البحوث الطبية وبسبب الرواتب القليلة جدا للأطباء ضعف البحث العلمي كثيرا وبرغم التحسن ما زال راتب الطبيب لا يتجاوز دولارا في الشهر فكيف تنمو الأبحاث أو يندفع في الخدمة ناهيك من البحوث وتطوير الحالة العلمية المادية يكفي أن نضرب مثلا خريجو إحدى السنين من الأطباء طالبا

Figure 5.3: Arabic Text After Cleansing

5.2.2 Word Tokeniser Module

For the purpose of Arabic word roots extraction, it is necessary to interpret the text as a sequence of words (tokens). A token here is defined as an Arabic string separated by whitespace (space, enter, and tab characters). The process of splitting an input text document into tokens is known as tokenisation. In natural language processing, tokenisation can occur at a number of different levels, where a text could be broken up into paragraphs, sentences, words, syllables, or phonemes. We are interested here in word tokenisation level, since a word form is the elementary unit to be stemmed. The author implemented

a program for the purpose of this module. This program distinguishes and breaks an input Arabic text into words. Figure 5.4 shows the text presented in Figure 5.3 after tokenisation.

المؤسسات	عليه	كثيرا	الطبية	راتب	الخدمة	إحدى
والمصانع	الآن	الأجهزة	وبسبب	الطبيب	ناهيك	السنين
التي	أن	الطبية	الرواتب	لا	من	من
تنتج	العالم	الحديثة	القليلة	يتجاوز	البحوث	الأطباء
أجهزة	أصبح	والمتطورة	جدا	دولارا	وتطوير	طالبا
ذات	قرية	ذات	للأطباء	في	الحالة	
علاقة	صغيرة	التقنيات	ضعف	الشهر	العلمية	
بالأبحاث	وبالرغم	الدقيقة	البحث	فكيف	المادية	
مراكز	من	والتي	العلمي	تنمو	يكفي	
البحوث	ذلك	بدونها	كثيراوبرغم	الأبحاث	أن	
الطبية	فإننا	لا	التحسن	أو	نضرب	
من	بحاجة	تثمر	ما	يندفع	مثلا	
المتفق	وتنقصنا	البحوث	زال	في	خريجو	

Figure 5.4: Arabic Text After Tokenising

5.2.3 Stopwords Removal Module

Stopwords are defined as the most frequent words in a corpus which do not convey any particular meaning of the document content. Stopwords have no roots, they consist of prepositions, conjunctions, adverbs and the like. For example, in English language, articles (e.g., a, an, the), prepositions (e.g., of, in, for, through), and pronouns (e.g., it, their, his) are considered as stopwords. Table 5.1 shows some examples of stopwords in Arabic language.

In this module, all tokens which were extracted from the previous module are checked against a predefined list of stopwords. If the token does have a match in this list, it will be excluded from any further processing; otherwise, it will be passed to the next module. The author compiled a list of Arabic stopwords, after an Arabic linguist has been consulted during this stage[1]. This procedure

[1]The author gratefully acknowledges the consultation and the assistance of Fahed Ashour; email: fahed.ashour@gsaa.uni-halle.de. Martin Luther University, Halle, Germany.

Stopword	Transli-teration	Stopword	Transli-teration	Stopword	Transli-teration
له	lh	منذ	mnḋ	إنك	Ạnk
أيها	ĀyhA	لهذه	lhḋh	منكن	mnkn
من	mn	لهذا	lhḋA	أنكم	Ānkm
هم	hm	للذان	llḋAn	منكم	mnkm
لأن	lĀn	لهذان	lhḋAn	ممكن	mmkn
لهن	lhn	للذين	llḋyn	ففيهما	ffyhmA
لهم	lhm	اللواتي	AllwAty	أنكما	ĀnkmA
وقد	wqd	هي	hy	نحن	nHn
أين	Āyn	أيهن	Āyhn	أيهم	Āyhm

Table 5.1: Example of Arabic Stopwords

is done as follows:

1. Compute the frequency of each word in the dataset[2].
2. Sort the words in the descending order based on their frequencies.
3. Compile the stopwords list by collecting the most frequently occurring words, in consultation with an Arabic linguist.

The set of this predefined stopwords contains a total of **1,237** stopwords. Appendix C shows these stopwords. Figure 5.5 shows the text presented in Figure 5.4 after the removal of the stopwords.

المؤسسات الطبية كثيرا البحوث العلمي تنمو العلمية
والمصانع المتفق الأجهزة الطبية كثيراوبرغم الأبحاث المادية
تنتج العالم الطبية وبسبب التحسن يندفع يكفي
أجهزة قرية الحديثة الرواتب راتب الخدمة نضرب
علاقة صغيرة والمتطورة القليلة الطبيب ناهيك خريجو
بالأبحاث مزدهرة التقنيات للأطباء يتجاوز البحوث الأطباء
مراكز بحاجة الدقيقة ضعف دولارا وتطوير طالبا
البحوث وتنقصنا تثمر البحث الشهر الحالة

Figure 5.5: Arabic Text After Stopwords Removal

Comparing the two texts in Figure 5.5 and Figure 5.4 respectively, it is clear

[2]The dataset used here is discussed in Chapter 6.

that the stopwords removal module successfully removed **29** stopwords found in the text in Figure 5.5.

5.2.4 Normalisation Module

According to Syiam et al. [135], normalisation is the process of unification of different forms of the same letter. To normalise the input word, the following steps are taken:

1. Break down the word into its compound letters. More formally, Let **W** is the input Arabic word, then the output is $W = (L_1, L_2, \ldots, L_n)$, where n is the length of W.
2. If $L_1 \in \{\text{أ}, \text{إ}, \text{آ}\}$ then $L_1 = \{\text{ا}\}$.
3. If $L_n = \{\text{ى}\}$ then $L_n = \{\text{ي}\}$.
4. If $L_i \in \{\text{ص}, \text{ض}, \text{ط}, \text{ظ}\}$ AND $L_{i+1} = \{\text{ط}\}$ OR
   $L_i \in \{\text{د}, \text{ذ}, \text{ز}\}$ AND $L_{i+1} = \{\text{د}\}$ Then $L_{i+1} = \{\text{ّ}\}$

The reason behind this normalisation is as follows:

In Step 2, The letters: [أ][Hamza above Alef], [إ][Hamza under Alef] and [آ][Alef Madah] should changed to the [ى][Plain Alef], since many people do not formally write the appropriate [Alef] at the beginning of the word. In Step 3, we replaced the letter [ى][Alef Maqsura] if it is the last letter in the word to the letter [ي], the reason for this being what we mentioned in Step 2.

Finally in Step 4, we came to an important aspect of the morpho-phonological approach in Arabic language, which is called mutation [الإبدال]. Mutation or morphological substitution can be defined as the changing or removing of a consonant letter and it is replacement with another. This is due to the difficulty in pronouncation. To clearify the concept of mutation, assume we want to derive an Arabic word [اصطبر](ASTbr,"long suffering") from the root [صبر](Sbr,"to be patient") using the morphological pattern [افتعل] (Aft'l) $AC_1tC_2C_3$. When the root [صبر](Sbr) is mapped to the pattern [افتعل], the resulted word is [اصتبر](AStbr). Morphologically, the third letter should be

changed from [ت](t) (which is a part of the pattern) to [ط](T) (which is the correct letter that should take a place in the word). More examples for mutation are presented in Figure 5.6. While, Figure 5.7 shows the text presented in Figure 5.5 after normalisation.

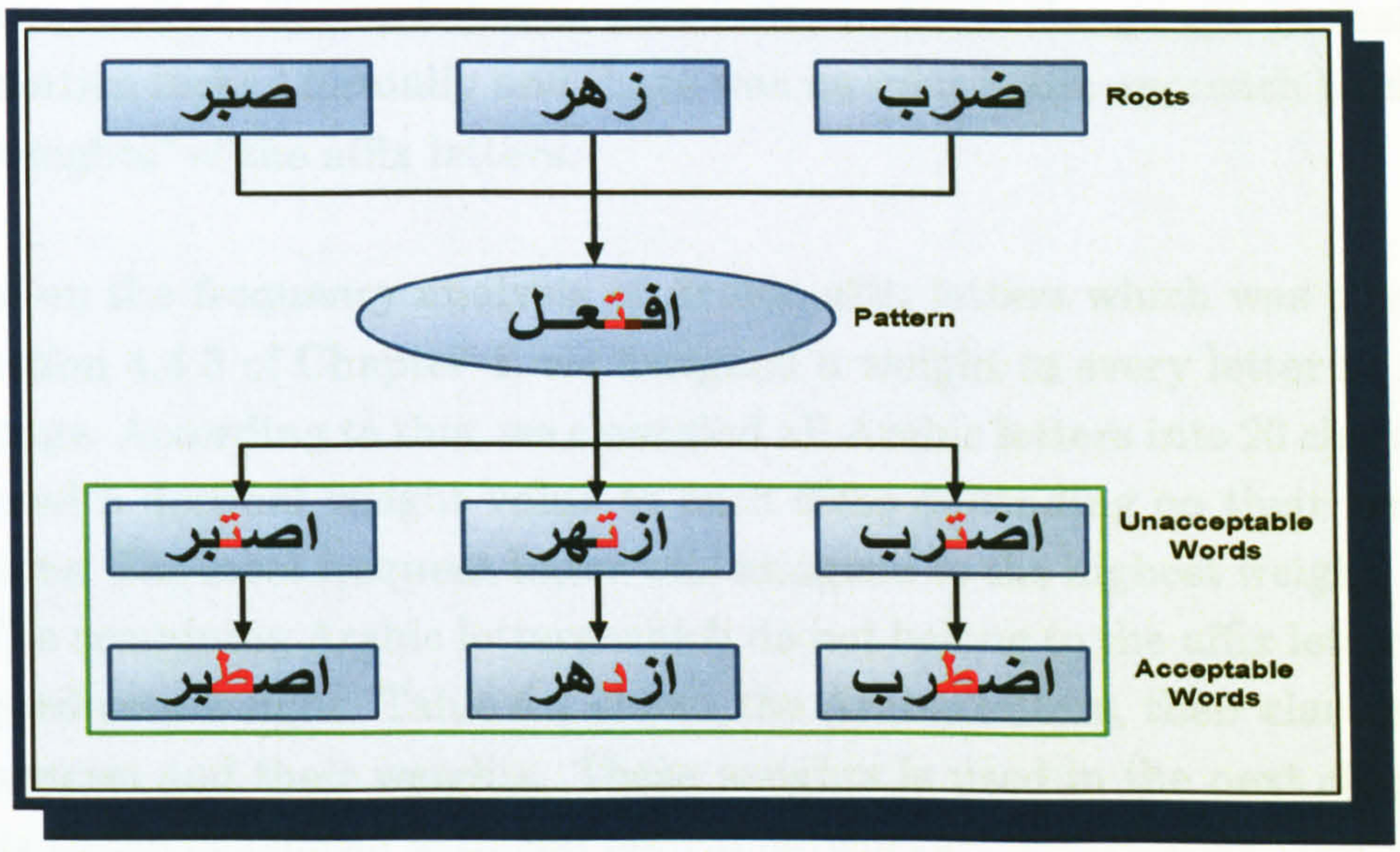


Figure 5.6: Mutation in Arabic Language

المؤسسات	الطبية	كثيرا	البحوث	العلمي	تنمو	العلمية
والمصانع	المتفق	الأجهزة	الطبية	كثيراوبرغم	الأبحاث	المادية
تنتج	العالم	الطبية	وبسبب	التحسن	يندفع	يكفي
اجهزة	قرية	الحديثة	الرواتب	راتب	الخدمة	نضرب
علاقة	صغيرة	والمتطورة	القليلة	الطبيب	ناهيك	خريجو
بالأبحاث	مزتهرة	التقنيات	للأطباء	يتجاوز	البحوث	الأطباء
مراكز	بحاجة	الدقيقة	ضعف	دولارا	وتطوير	طالبا
البحوث	وتنقصنا	تثمر	البحث	الشهر	الحالة	

Figure 5.7: Arabic Text After Normalisation

Note that the word [**أجهزة**] is normalised to [**اجهزة**], and the word [**مزدهرة**] is normalised to [**مزتهرة**].

5.2.5 Weights Module

The developed computational model presented in this study deals only with numeric input data. Therefore, the raw data (words) must be converted from Arabic letters to numeral equivalents. Earlier work by the author [25] showed some numerical relation between affix letter in Arabic language. In that work the relation lacked formally and there was no systematic approach to identify the "weights" of the affix letters.

Based on the frequency analysis of Arabic affix letters which was presented in Section 4.4.3 of Chapter 4, we assigned a weight to every letter in Arabic language. According to this, we classified all Arabic letters into 20 classes and assigned a decimal weight value to each class depending on their term frequencies. The most frequent letter was assigned to the highest weight, and so on. The remaining Arabic letters which do not belong to the affix letters were assigned zero weight. Table 5.2 shows the Arabic letters, their classes, their frequencies and their weights. These weights is used in the next chapter as inputs to the developed artificial neural networks.

Figure 5.8 and Figure 5.9 show two examples for the two encoded Arabic words. The first Figure for the Arabic word [والمصانع] (wAlmSAn',"and the factories"), while the second Figure for the Arabic word [مزدهرة] (mzdhrt). Note that in the second example, the third letter [د](d) is replaced with [ت](t) due mutation.

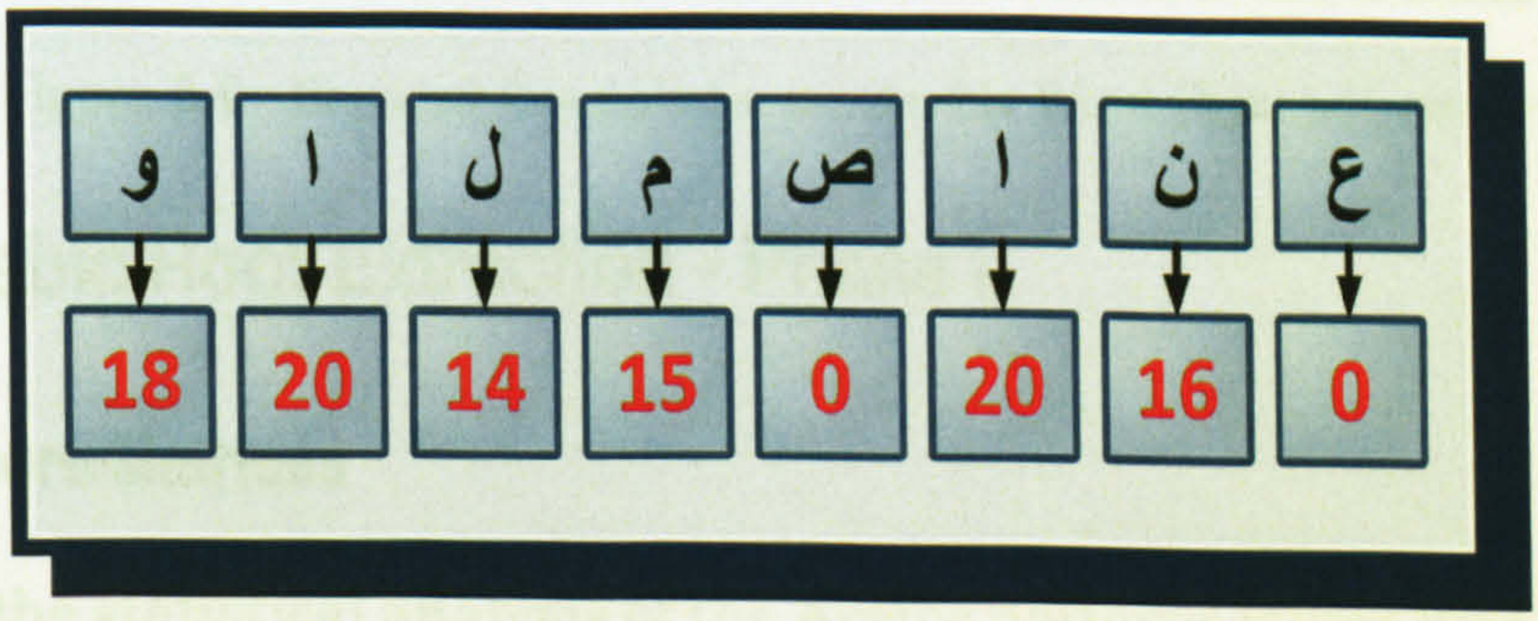


Figure 5.8: Encoded Example for the Arabic Word [والمصانع](*wAlmSAn'*)

Letter	Transliteration	Frequency	Weight
ا	A	66,862	20
ي	y	24,694	19
و	w	23,167	18
ت	t	21,025	17
ن	n	19,170	16
م	m	12,156	15
ل	l	9,688	14
ه	h	9,501	13
ة	T	9,320	12
إ	b	4,558	11
س	s	1,941	10
أ	k	1,704	9
ى	y	1,623	8
ئ	Ã	1,527	7
ء	’	1,467	6
ؤ	w̄	1,403	5
آ	A	1,351	4
ف	f	1,345	3
ب	b	1,212	2
ك	k	1,193	1
باقي الحروف	Other Letters		0

Table 5.2: Arabic Letter Classifications

Figure 5.9: Encoded Example for the Arabic Word [مزدهرة](*mzdhrt*)

5.3 Arabic Root Extraction - Phase I

5.3.1 Score Matrices

Based on the statistical analysis of the Arabic patterns which was discussed in Chapter 4, **nine** score matrices are generated. Every pattern category length has its own score matrix. We denoted these categories as: $patt_4$ for pattern

category of length **4**, $patt_5$ for pattern category of length **5**, and so on (see Section 4.3 of Chapter 4).

The score matrices were built up from the possibility of occurrence of each letter as an affix letter. The reason behind these generations is that the set of the affix letters may vary from pattern category to another as it was discussed in Chapter 4. For example the letter [س](s) is considered as an affix letter in $patt_5$, while it is not in $patt_4$. More deeply, the letter [س](s) is considered as an affix letter in $patt_5$ just in position **1**, while it is not in the other positions. Figures 5.10 and 5.11 clear these results. In conclusion, the score matrices unfold more details about the word letters and unveil the root letters in an obvious form.

To know how score matrix works, assume we have four affix letters {A, B, C, D} and a pattern category with length 4. The score matrix of this assumption is as follows:

$$\mathbf{P} = \begin{array}{c} \\ \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The above matrix has row labels {A, B, C, D} for the affix letters set, and column labels {1, 2, 3, 4} for the letter positions. That is, P_{11} is a score value refers to the entry at the first row (letter A) and the first column (position 1). In general, P_{ij} refers to the entry at the i^{th} row (affix letter) and the j^{th} column (letter position). If $P_{ij} = 1$, then this means that the letter i is considered as an affix (non-root) letter in position j. Otherwise it is considered as a root-letter in this position.

As mentioned above, we generated **9** score matrices SM_4 through SM_{12}, with one score matrix for every pattern category length, i.e., the score matrix SM_4 for pattern category $patt_4$, SM_5 for pattern category $patt_5$, and so on. Fig-

ure 5.10 and Figure 5.11 show the score matrices for the pattern categories SM_4 and SM_5, respectively. The complete score matrices for all pattern categories are illustrated in Appendix D.

	1	2	3	4
[ا](A)	1	1	1	1
[ي](y)	1	1	1	1
[و](w)	1	1	1	1
[ت](t)	1	0	0	1
[ن](n)	1	0	0	1
[م](m)	1	0	0	0
[ل](l)	1	0	0	0
[ه](h)	0	0	0	1
[ة](t)	0	0	0	1
[إ](Ạ)	0	0	0	0
[س](s)	0	0	0	0
[أ](Ā)	0	0	0	0
[ى](A)	0	0	0	1
[ئ](Ã)	0	0	0	0
[ء](')	0	0	0	0
[ؤ](w̄)	0	0	0	0
[آ](A)	0	0	0	0
[ف](f)	1	0	0	0
[ب](b)	1	0	0	0
[ك](k)	1	0	0	1

Figure 5.10: Score Matrix SM_4 for $patt_4$

	1	2	3	4	5
[ا](A)	1	1	1	1	1
[ي](y)	1	1	1	1	1
[و](w)	1	1	1	1	1
[ت](t)	1	1	1	1	1
[ن](n)	1	1	0	1	1
[م](m)	1	1	0	0	1
[ل](l)	1	1	0	0	0
[ه](h)	0	0	0	1	1
[ة](t)	0	0	0	0	1
[إ](Ạ)	0	0	0	0	0
[س](s)	1	0	0	0	0
[أ](Ā)	0	0	0	0	0
[ى](A)	0	0	0	0	1
[ئ](Ã)	0	0	0	1	0
[ء](')	0	0	0	0	1
[ؤ](w̄)	0	0	0	0	0
[آ](A)	0	0	0	0	0
[ف](f)	1	0	0	0	0
[ب](b)	1	1	0	0	0
[ك](k)	1	0	0	1	1

Figure 5.11: Score Matrix SM_5 for $patt_5$

For example, the letter [م](m) in SM_4 (Figure 5.10) has the score values [1, 0, 0, 0]. This means that the letter is considered as a **root-letter**, if it appears in any of the positions 2, 3, and 4 in any given Arabic word with length is 4. While, if it is on the first position, it may be a **root-letter** or it may be not. Figure 5.12 shows five word examples have a letter [م](m) in different positions.

In Figure 5.12 (a) the letter [م](m) is in the first position and it is not a root-letter, while in (b) it is. In (c), (d), and (e) the letter is considered as a **root-letter**. This is follow the definition of the score values of [م](m) in SM_4.

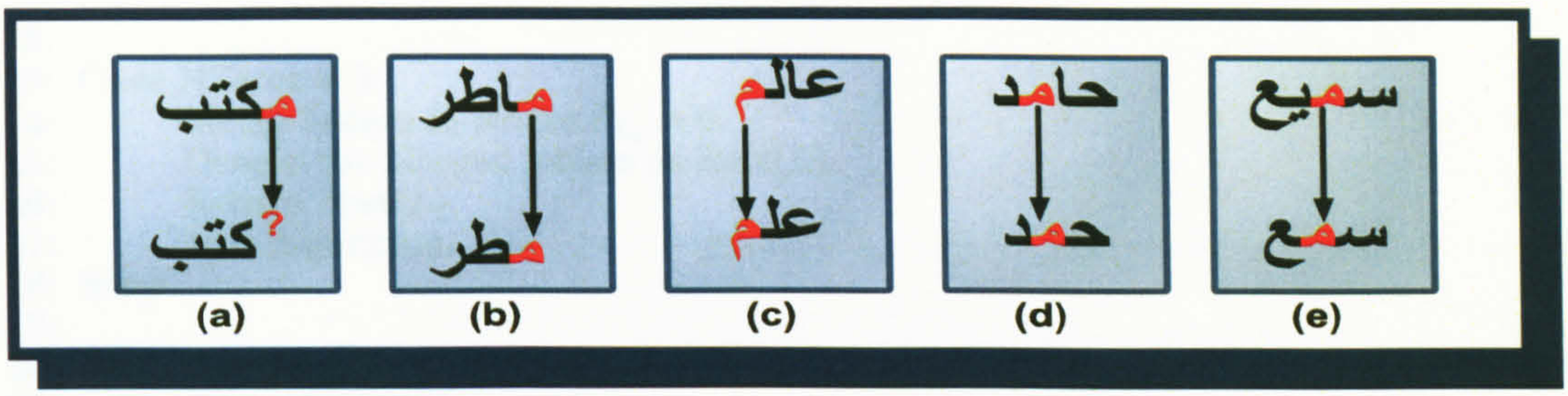


Figure 5.12: The Letter [م](*m*) in Different Positions

5.3.2 Root Extraction Phase I

As mentioned eariler, **Phase I** tries to extract the root of a given word by exploiting the relations between word letters. In the present work, a novel algorithm was developed, with the aim of achieving this goal. The algorithm is based on the weight and score values that was presented in the previous sections of this chapter. The pseudo code of the algorithm is presented in Algorithm 5.1. Some examples are presented on the next pages to illustrate the way the algorithm works. Two examples are presented for each case.

ALGORITHM 5.1 : Pseudo Code of the Developed Stemmer (Phase I)

INPUT
- Let **L** denotes an Arabic word with length = **n**, $L = \{L_1, L_2, \ldots, L_n\}$;
- Let **W** denotes the weights of **L**, $W = \{W_{L_1}, W_{L_2}, \ldots, W_{L_n}\}$;
- Let **SM** denotes the score values of **L**, $SM = \{SM_{L_1}, SM_{L_2}, \ldots, SM_{L_n}\}$.

OUTPUT
- The root of **L** denoted by **Root(L)**.

1: Initialise **NZeros**.
2: **for all** $i \leftarrow 1, n$ **do**
3: Compute $P_{L_i} = W_{L_i} * SM_{L_i}$
4: **if** $P_{L_i} = 0$ **then**
5: **NZeros** ← **NZeros + 1**.
6: **end if**
7: **end for**
8: Select Case of (**NZeros**)
9: **Case** NZeros ≥ 4:
10: Exit Algorithm.
11: **Stop**

```
12:
13: Case NZeros = 3:
14:         Merge letters L_i where P_{L_i} = 0.
15:         Denote the Merged letters as Root(L).
16:         Return Root(L).
17:         Exit Algorithm.
18: Stop
19:
20: Case NZeros = 2:
21:         N_1 ← the position of 1st zero.
22:         N_2 ← the position of 2nd zero.
23:         Root−Distance ← N_2 - N_1 - 1.
24:         if Root−Distance ≥ 2 then
25:             Merge letters L_i where P_{L_i} = 0; and
26:             the letter that has minimum weight occurs between N_1 and N_2.
27:             Denote the Merged letters as Root(L).
28:             Return Root(L).
29:             Exit Algorithm.
30:         else
31:             Extract the root using the trained neural networks.
32:         end if
33: Stop
34:
35: Case NZeros ≤ 1:
36:         Extract the root using the trained neural networks.
37: Stop
38:
39: End Selection
```

Example 1

Assume the tested word is [بالأبحاث] (bAlĀbHAt̂, "using the researchs"), this word has **8** letters, which are, L ={[ب](b), [ا](A), [ل](l), [آ](Ā), [ب](b), [ح](H), [ا](A), [ث](t̂)}. The corresponding weight values are **W** =[2, 20, 14, 9, 2, 0, 20, 0].

The process of determining the score values for each letter is started. These score values are extracted from score matrix SM_8 (see Appendix D) since the length of the tested word is **8**. The score value for the first letter [ب](b) is **1**, for the second letter [ا](A) is also **1**, and so on. The score values of all letters in the tested word are **SM** =[1, 1, 1, 1, 0, 0, 1, 0].

In Step 3, a new vector P_L is generated by multiplying vector **W** by vector **SM**. The values of this vector are P_L =[2, 20, 14, 9, 0, 0, 20, 0]. In Step 5, the process continues counting the number of zeros in the generated vector P_L.

In the current example, there are three zeros, which are in the positions **5**, **6**, and **8**. The zeros mean that the corresponding letters are a **root-letters**. So, the case-statement in Step 13 is triggered. In Steps 14 and 15, the letters {[ب](b), [ح](H), [ث](t̂)}, which are corresponding to the zero values are merged. In Step 16, the extracted root [بحث](bHt̂,"search") is returned, and this is the correct root for the tested word.

All the previous steps for the current example are summarised in Table 5.3. The root columns are shaded by yellow. Table 5.4 presents another tested word of the same type as the word that is presented in Table 5.3.

Input Word: بالأبحاث												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i					ث	ا	ح	ب	أ	ل	ا	ب
W_i					0	20	0	2	9	14	20	2
SM_i					0	1	0	0	1	1	1	1
P_i					0	20	0	0	9	14	20	2
Output Root: بحث												

Table 5.3: Extraction of the Root for the Word [بالأبحاث] using Phase I

Input Word: بالدراستين												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i			ن	ي	ت	س	ا	ر	د	ل	ا	ب
W_i			16	19	17	10	20	0	0	14	20	2
SM_i			1	1	1	0	1	0	0	1	1	1
P_i			16	19	17	0	20	0	0	14	20	2
Output Root: درس												

Table 5.4: Extraction of the Root for the Word [بالدراستين] using Phase I

Example 2

The second tested example is the Arabic word [كالجامعات] (wAlmSAn',"as the universities"). The length of this word is **9**. The weight values are **W**=[1, 20, 14, 0, 20, 15, 0, 20, 17]. The score values are **SM**=[1, 1, 1, 0, 1, 1, 0, 1, 1]. The generated multiplying values are P_L=[1, 20, 14, 0, 20, 15, 0, 20, 17].

It is clear that the vector P_L has only two zeros in the positions 4 and 7, respectively. In this case, as the Step 20 suggests, the **root-distance** between these positions is computed (Step 23). The value here is **2 (=7 - 4 - 1)**, which is satisfied the condition in Step 24 (**Root-Distance** $\geq$ **2**). This means that the third root-letter is the one that is in the position 5 or 6, since the other root-letters are in the positions 4 and 7, respectively. The problem now how to choose the correct position for the third root-letter. Is it in the position 5 or 6 ?.

Based on the analysis which has been done on Chapter 4, we found that the letter which has the lowest weight and is occupy between two root-letters is strongly candidate to be the root-letter. This procedure seems gave the correct root-letter all the time. For this reason we assigned the weight values to the Arabic letters in a descending order in terms with their frequencies in appearing as affix letters.

In the current example, the letter in the position **5** has a weight **20**, and the letter in the position **6** has a weight **15**. The lowest weight is for the letter in position **6**. As a consequence, this letter [م](m) is chosen as a root-letter. The unchosen letter is the letter [ا](A) which is a vowel letter. Now, the root is generated by merging the three letters {[ج](j), [م](m), [ع](')} in the positions **4**, **6**, and **7**, respectively. The output root is [جمع] (jm',"sum"), and this is the correct root for the tested word. Table 5.5 summarises all the previous steps, while Table 5.6 presents another tested word of the same type as the word that is presented in Table 5.5.

Input Word: كالجامعات												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i				ت	ا	ع	م	ا	ج	ل	ا	ك
W_i				17	20	0	15	20	0	14	20	1
SM_i				1	1	0	1	1	0	1	1	1
P_i				17	20	0	15	20	0	14	20	1
Output Root: جمع												

Table 5.5: Extraction of the Root for the Word [كالجامعات] using Phase I

Input Word: شواهد												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i								د	هـ	ا	و	ش
W_i								0	13	20	18	0
SM_i								0	1	1	1	0
P_i								0	13	20	18	0
Output Root: شهد												

Table 5.6: Extraction of the Root for the Word [شواهد] using Phase I

Example 3

The third tested example is the Arabic word [معلمون] (m'lmwn,"teachers"). The length of this word is **6**. The weight values are **W**=[15, 0, 14, 15, 18, 16]. The score values are **SM**=[1, 0, 1, 0, 1, 1]. The generated multiplying values are P_L=[15, 0, 14, 0, 18, 16].

P_L has only two zeros in the positions 2 and 4, respectively, and this is met the condition in Step 20. According to Step 23, the **Root-Distance** between these positions is computed. The value here is **1(=4 - 2 - 1)**, which is not satisfied the condition in Step 24 (**Root-Distance** $\geq$ **2**). Therefore, **Phase I** can not deal with this word. In consequence, **Phase I** passes the word to **Phase II** in order to check it against one of the trained artificial neural networks. **Phase II** is presented in Chapter 6. Table 5.6 summarises the computation values of the current example, while Table 5.8 presents another tested word of the same type as the word that is presented in Table 5.6.

Input Word: معلمون												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i							ن	و	م	ل	ع	م
W_i							16	18	15	14	0	15
SM_i							1	1	0	1	0	1
P_i							16	18	0	14	0	15
Output Root: Goto Phase II												

Table 5.7: Extraction of the Root for the Word [معلمون] using Phase I

Input Word: يستعملونها												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i			ا	هـ	ن	و	ل	م	ع	ت	س	ي
W_i			20	13	16	18	14	15	0	17	10	19
SM_i			1	1	1	1	0	1	0	1	1	1
P_i			20	13	16	18	0	15	0	17	10	19
Output Root: Goto Phase II												

Table 5.8: Extraction of the Root for the Word [يستعملونها] using Phase I

Example 4

The forth tested example is the Arabic word [مواهب] (mwAhb,"talents"). The length of this word is **5**. The weight values are **W**=[15, 18, 20, 13, 2]. The score values are **SM**=[1, 1, 1, 1, 0]. The generated multiplying values are P_L=[15, 18, 20, 13, 0].

It is clear that the vector P_L has only one zero in the last position. In this case, Step 35 is triggered, and the current tested word is passed to **Phase II** in order to process it using artificial neural networks. Table 5.9 presents this example, while Table 5.10 illustrates another word which has no zeros in P_L.

Input Word: مواهب												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i								ب	هـ	ا	و	م
W_i								2	13	20	18	15
SM_i								0	1	1	1	1
P_i								0	13	20	18	15
Output Root: Goto Phase II												

Table 5.9: Extraction of the Root for the Word [مواهب] using Phase I

Input Word: تملكها												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i							ا	هـ	ك	ل	م	ت
W_i							20	13	1	14	15	17
SM_i							1	1	1	1	1	1
P_i							20	13	1	14	15	17
Output Root: Goto Phase II												

Table 5.10: Extraction of the Root for the Word [تملكها] using Phase I

Example 5

The last tested example is the word [الديمقراطية] (AldymqrATyt, "democracy"). The length of this word is 11. The weight values are W=[20, 14, 0, 19, 15, 0, 0, 20, 0, 19, 12]. The score values are SM=[1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1]. The generated multiplying values are P_L=[20, 14, 0, 0, 15, 0, 0, 20, 0, 19, 12].

The vector P_L has five zeros in the positions 3, 4, 6, 7, and 9. This satisfies the condition (NZero $\geq$ 4) in Step 9. In this case, no further process is carried out. The algorithm terminates its processing and the input word is returned without stemming. The reason behind this, is that the triliteral root has only three letters, which means that the vector P_L should have at most three zeros.

In the cases of four or five zeros, the tested word could be a valid Arabic word and has a root rather than triliteral one. But, in the case where there are more than five zeros, the tested word is definitely classified as an Arabised word (borrowed word). In this way, our approach can assist in extracting the borrowed words found in an Arabic text.

Table 5.11 presented the current tested word and its computation vectors. This word is not an Arabic word, it is borrowed from Latin to present the meaning of "democracy".

Input Word: الديمقراطية												
i	12	11	10	9	8	7	6	5	4	3	2	1
L_i		ة	ي	ط	ا	ر	ق	م	ي	د	ل	ا
W_i		12	19	0	20	0	0	15	19	0	14	20
SM_i		1	1	0	1	0	0	1	0	0	1	1
P_i		12	19	0	20	0	0	15	0	0	14	20
Output Root: No Stemming												

Table 5.11: Extraction of the Root for the Word [الديمقراطية] using Phase I

The dataset used in this study, which is discussed at length in Chapter 6, shows that **42,714** from **66,344** type words are satisfied the pre-conditions of the **Phase I** processed in it, while the remaining **23,630** type words are passed to the **Phase II**. The evaluation of the **Phase I** is presented in Chapter 7.

5.4 Summary

This chapter has introduced the new computational model for extraction of the Arabic word roots. An overview was given about the system architecture and its modules. The model consists of four supplementary modules, in addition to the two major modules: **Phase I** and **Phase II**. A detailed discussion was given for each module.

Phase I is presented in this chapter, while **Phase II** is presented in Chapter 6. **Phase I** tries to extract the root of a given word by exploiting the numerical relations between the word letters. To achieve this aim a novel algorithm was developed, and presented in this chapter. The algorithm is based on the weight and score values that assigned to the tested word letters. These values are extracted from the comprehensive analysis that was discussed in

the previous chapters. By the end of the chapter, some examples are presented in order to illustrate how the algorithm works.

The next chapter deals with the **Phase II** of the computational approach of the extraction of the Arabic word roots which is based on the artificial neural networks.

Chapter 6

ARABIC ROOT EXTRACTION USING NEURAL NETWORKS - PHASE II

6.1 Introduction

This chapter presents the **Phase II** of the MUAIDI-STEMMER algorithm, which deals with the words that are not processed in **Phase I** of Chapter 5. **Phase II** is based on artificial neural networks (ANNs) trained by the backpropagation learning rule. In this phase, the root extraction problem is formulated as a classification problem and the ANN as a classifier created by supervised learning.

This chapter starts by presenting the ANN methodology used in this study. Then the chapter discusses the dataset used in this research, and how it is prepared. After that, it shows the topologies for all the generated ANNs. And finally, this chapter presents the optimal parameters for all the ANNs.

6.2 Neural Network Methodology Used in this Research

As mentioned in the second part of the literature review of Chapter 3, ANNs are known for their ability to generalise and memorise according to the sim-

ilarity relations of their inputs. Furthermore, they have the ability to distinguish different output classes according to the inputs that are similar on their values. This generalisation grants ANNs the power to classify and identify the correct output class, after an adequate level of training, and even the input values they have never seen before. This nature of ANNs offer a promising solution to the problem of the extraction of roots. The novel solution reported here tries to let the ANNs to learn the relations between the input letters instead of preparing lookup tables, or a list of Arabic patterns as with conventional approaches. The modelling ANN solution presented in this chapter is based on the backpropagation learning rule [76].

Backpropagation neural networks (BPNNs) are nowadays widely used in many applications and have the ability to model very complex problems. One of the main drawbacks for BPNNs as such as other types of ANNs is to find the suitable network topology that generalises well on unseen data [49]. Also, another problem is that some applications might involve a large dataset, which leads to architecture complexity. In consequence, the training time could be increased.

In the literature of ANNs , almost all of the studies focus on finding a single ANN for a specific problem. Jiri Sima [129] stated that some ANNs appear very time consuming even for small network architectures. Also, he proved that BPNN is not efficient for large tasks, due to their architecture complexities. Moreover, many studies found that combining several ANNs, can enhance the generalisation of the whole system over the separate generalisation ability of the individuals.

For all the above reasons, in this study we decomposed the Arabic roots extraction complex task into a number of simple tasks. We assigned a specific BPNN for each task. The term task refers here to the Arabic words category that have the same length, i.e., the category of Arabic words with length four has its own BPNN [23][22], the category of Arabic words with length five has its own BPNN and so on. We denoted this approach as a multi-backpropagation neural network (MBPNN). This approach is proposed as

an alternative training method for BPNN to minimise the ANN complexity. Minimising ANN complexity means splitting the huge training data set into several smaller training data sets. In consequence, this could reduce the network learning time and increase its learning capability.

Eight BPNNs are generated as illustrated in Figure 6.1. Every BPNN has its own topology and has labelled as $BPNN_i$, where the index i refers to the length of the input word. Based on the analysis that was presented in Chapter 4, the value of i is ranged from 4 to 11. In **Phase II** any word with length = 12 is ignored, since all of these words are processed in **Phase I**. For any tested Arabic word, only one BPNN is triggered according to the word length.

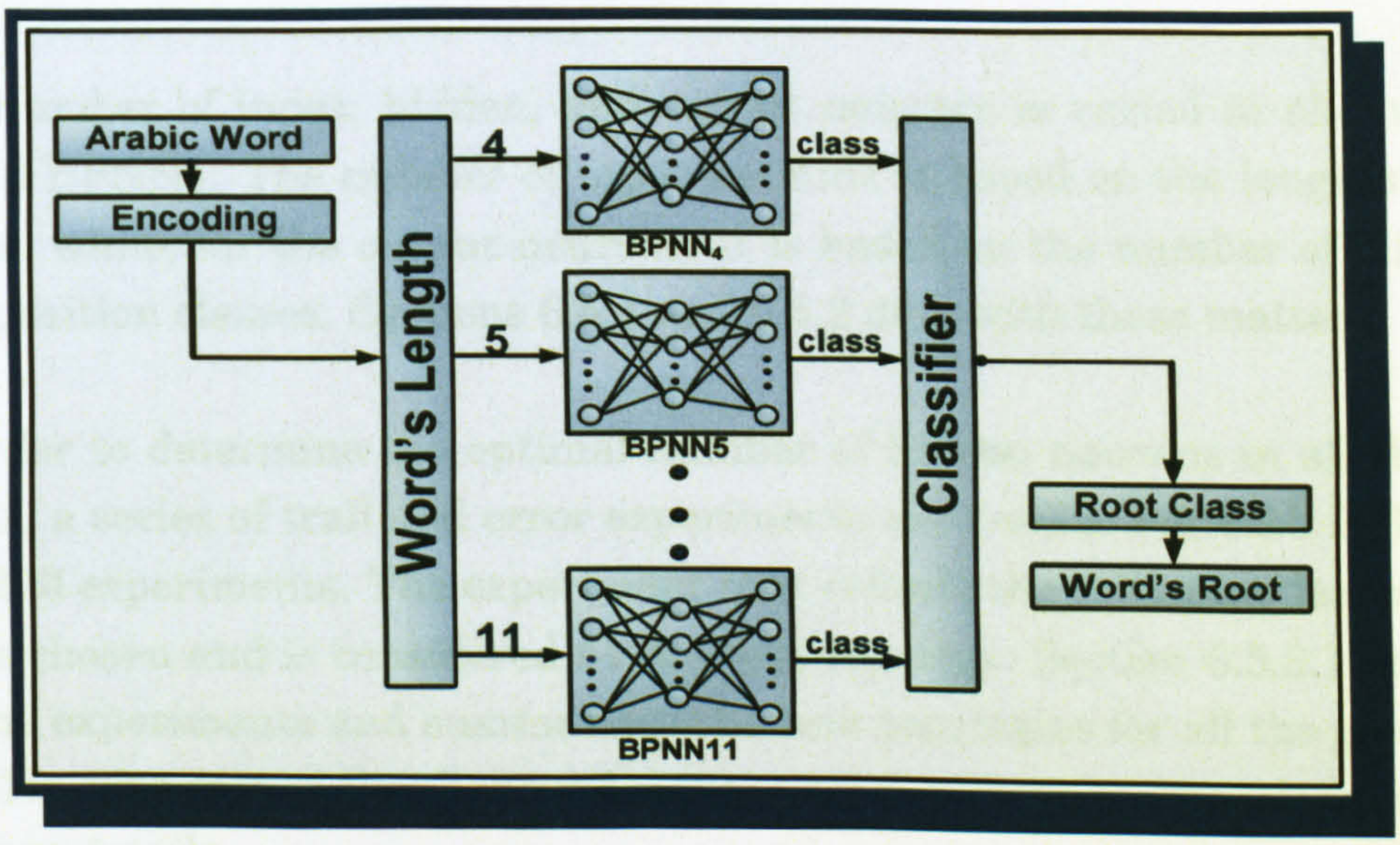


Figure 6.1: The Architecture of **Phase II** Based on MBPNNs

The dataset used in this study is drawn from [29]. Section 6.3 describes the dataset at length. Roughly **75%** of the dataset is used to train the BPNNs, and **25%** is used as a test dataset. These two datasets are chosen at random. The training dataset is utilised for optimising the BPNNs and for selecting their parameter values, whereas the test datasets is used solely in the final evaluation of the generated BPNNs (see Chapter 7).

All the generated BPNNs were trained using the standard backpropagation learning algorithm (Section 3.9.2 of Chapter 3). With this method, backpropagation is performed with the training set. The update rule in the method used here tries to minimise the squared error over the training set. The test data set is applied to the BPNNs in order to test how well the network will be able to generalise the data it has not trained on.

The architectures of the generated BPNNs are based on a multi-layer network architecture. The architecture has three layers: the input layer, one hidden layer and the output layer. All the neurons in these layers are fully connected. The activation function used in the hidden layer as well as the output node is a basic sigmoid activation function.

The number of input, hidden, and output neurons is varied in all the generated BPNNs. The number of input neurons is based on the lengths of the words, while, for the output neurons, it is based on the number of different root position classes. Sections 6.5.1 and 6.5.2 deal with these matters.

In order to determine the optimal number of hidden neurons in all the networks, a series of trail and error experiments are tested. For each BPNN we ran **200** experiments. The experiment that reflects the minimum learning error is chosen and is considered as the best topology. Section 6.5.3.1 presents all the experiments and summarises the best topologies for all the generated BPNNs. The rest of the chapter describes all the above methodology matters in more details.

6.3 Corpus

The word **corpus**[1] is used to refer to any text in written or spoken form [56]. In computational linguistics this term is used to refer to the large collections of machine-readable texts in a specific language (or language variety) [56].

[1]This term is derived from the latin word meaning "body".

There are two types of corpus, with respect to the information included. First, the **raw-text corpora**[2], in which the corpus contains raw text (plain text) with no additional information. Secondly, the **annotated corpus**, which involves the addition of analytical information to the plain text, such as morphological features (roots, stems, affixes) and part-of-speech tags. The existence of annotated corpus and its relevance to the research is a substantial factor in developing any natural language processing application. Also, a good sized corpus can show a significant language morphological behavior.

Unlike English, Arabic language suffers from the lack of sufficient resources in the field of corpora [32]. Furthermore, it suffers from the absence of a large reliable linguistic corpus for research purposes [69]. The scarce availability of an annotated Arabic corpus is one of the biggest challenges that faces the Arabic researchers in the computational linguistics discipline. There is a small number of available Arabic corpora; but unfortunately, most of them are not free. A good survey of the existing corpora resources for Arabic language can be found in [29].

The corpus used in this research was drawn from Latifa Al-Sulaiti's homepage [29] which is called **Corpus of Contemporary Arabic** (hereafter refer to as **CCA**). This free corpus was created by Latifa Al-Sulaiti with the cooperation of Eric Atwell as part of her MSc project at the University of Leeds in 2004 [30]. For our knowledge, this corpus is the only free one found in the field.

CCA contains electronically written full text of 431 articles[3], which were presented as XML format (Figure 6.2 shows an example). According to Latifa [30][31][32], the sources for these articles are: newspapers, magazines, and web pages. Newspaper texts and web pages are the most accessible source of modem standard Arabic.

[2]Corpora is the plural of corpus.
[3]Also, **CCA** contains some spoken Arabic texts.

There are two types of corpus, with respect to the information included. First, the **raw-text corpora**[2], in which the corpus contains raw text (plain text) with no additional information. Secondly, the **annotated corpus**, which involves the addition of analytical information to the plain text, such as morphological features (roots, stems, affixes) and part-of-speech tags. The existence of annotated corpus and its relevance to the research is a substantial factor in developing any natural language processing application. Also, a good sized corpus can show a significant language morphological behavior.

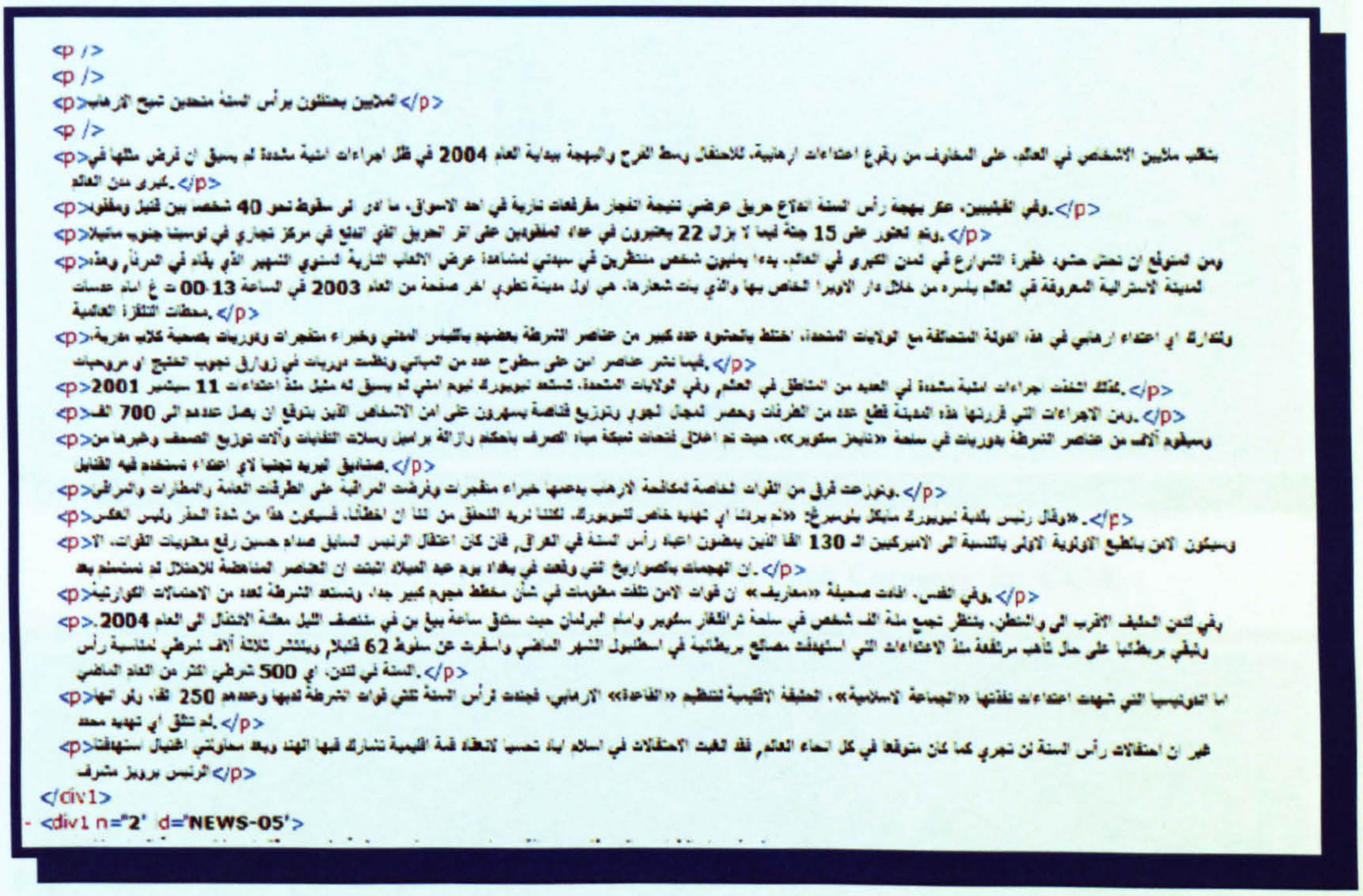
Unlike English, Arabic language suffers from the lack of sufficient resources in the field of corpora [32]. Furthermore, it suffers from the absence of a large reliable linguistic corpus for research purposes [69]. The scarce availability of an annotated Arabic corpus is one of the biggest challenges that faces the Arabic researchers in the computational linguistics discipline. There is a small number of available Arabic corpora; but unfortunately, most of them are not free. A good survey of the existing corpora resources for Arabic language can be found in [29].

The corpus used in this research was drawn from Latifa Al-Sulaiti's homepage [29] which is called **Corpus of Contemporary Arabic** (hereafter refer to as **CCA**). This free corpus was created by Latifa Al-Sulaiti with the cooperation of Eric Atwell as part of her MSc project at the University of Leeds in 2004 [30]. For our knowledge, this corpus is the only free one found in the field.

CCA contains electronically written full text of 431 articles[3], which were presented as XML format (Figure 6.2 shows an example). According to Latifa [30][31][32], the sources for these articles are: newspapers, magazines, and web pages. Newspaper texts and web pages are the most accessible source of modem standard Arabic.

[2]Corpora is the plural of corpus.
[3]Also, **CCA** contains some spoken Arabic texts.

The articles in **CCA** covered 15 categories of the human knowledge in Arabic, such as, autobiography, children's stories, economics, education, health and medicine, interviews, politics, religion, science, short stories, sociology, sports, and tourist and travel. Figure 6.3 shows the number of tokens in each category, while, Figure 6.4 shows the number of word-types in each category.



```
<p />
<p />
<p>الملايين يحتفلون برأس السنة متحدين شبح الارهاب</p>
<p />
<p>بتغلب ملايين الاشخاص في العالم، على المخاوف من وقوع اعتداءات ارهابية، للاحتفال وسط الفرح والبهجة ببداية العام 2004 في ظل اجراءات امنية مشددة لم يسبق ان فرض مثلها في كبرى مدن العالم.</p>
<p>وفي الفيليبين، عكر بهجة رأس السنة انفجار هريق عرضي نتيجة انفجار مفرقعات نارية في احد الاسواق، ما ادى الى سقوط نحو 40 شخصا بين قتيل ومفقود.</p>
<p>ويتم العثور على 15 جثة فيما لا يزال 22 يعتبرون في عداد المفقودين على اثر الحريق الذي اندلع في مركز تجاري في توبيتا جنوب مانيلا.</p>
<p>ومن المتوقع ان تحتل حشود غفيرة الشوارع في لندن الكبرى في العالم، بدءا بمليون شخص منتظرين في سيدني لمشاهدة عرض الالعاب النارية لسنوي الشهير الذي يقام في الثرثار وهذه المدينة الاسترالية المعروفة في العالم بأسره من خلال دار الاوبرا الخاص بها والذي بات شعارها، هي اول مدينة تطوي اخر صفحة من العام 2003 في الساعة 13.00 ت غ امام عدسات محطات التلفزة العالمية.</p>
<p>ولمكافحة اي اعتداء ارهابي في هذه الدولة المتحالفة مع الولايات المتحدة، اختلط بالمحتفلين عدد كبير من عناصر الشرطة بعضهم بالثياب المدني وخبراء متفجرات ودوريات بصحبة كلاب مدربة فيما انتشر عناصر امن على سطوح عدد من المباني ونظمت دوريات في زوارق نجوب الخليج او مروحيات.</p>
<p>هذا وقد اتخذت اجراءات امنية مشددة في العديد من المناطق في العالم وفي الولايات المتحدة، تستعد نيويورك ليوم امني لم يسبق له مثيل منذ اعتداءات 11 سبتمبر 2001.</p>
<p>ومن الاجراءات التي قررتها هذه المدينة قطع عدد من الطرقات وحصر المجال الجوي وتوزيع قناصة يسهرون على امن الاشخاص الذين يتوقع ان يصل عددهم الى 700 الف.</p>
<p>وسيقوم آلاف من عناصر الشرطة بدوريات في ساحة «تايمز سكوير»، حيث تم اغلاق فتحات شبكة مياه الصرف باحكام وازالة براميل وسلات النفايات وآلات توزيع الصحف وغيرها من مخابئ قنابل محتملة.</p>
<p>وخوزعت فرق من القوات الخاصة لمكافحة الارهاب بدعمها خبراء متفجرات وفرضت المراقبة على الطرقات العامة والمطارات والمرافئ.</p>
<p>«وقال رئيس بلدية نيويورك مايكل بلومبرغ: «لم يرانا اي تهديد خاص لنيويورك، لكننا نريد التحقق من اننا احطنا. فسيكون هذا من شدة الحذر وليس العكس.</p>
<p>وسيكون الامن بالقطع الاولوية الاولى بالنسبة الى الاميركيين الـ 130 الفا الذين يمضون اعياد رأس السنة في العراق. فان كان اعتقال الرئيس السابق صدام حسين رفع معنويات القوات، الا ان الهجمات بالصواريخ التي وقعت في بغداد يوم عيد الميلاد اثبتت ان العناصر المناهضة للاحتلال لم تستسلم بعد.</p>
<p>وفي الفلبين، اذاعت صحيفة «معاريف» ان قوات الامن تلقت معلومات في شأن مخطط مجرم كبير جدا، وتستعد الشرطة لعدد من الاحتمالات الكوارثية.</p>
<p>وفي لندن المليئة الاقرب الى والمتطرف، ينتظر تجمع مئة الف شخص في ساحة ترافالغار سكوير واسم البرلمان حيث ستدق ساعة بيغ بن في منتصف الليل معلنة الانتقال الى العام 2004.</p>
<p>وتبقي بريطانيا على حال تأهب مرتفعة منذ الاعتداءات التي استهدفت مصالح بريطانية في اسطنبول الشهر الماضي واسفرت عن سقوط 62 قتيلا ويتشر ثلاثة آلاف شرطي لمناسبة رأس السنة في لندن، اي 500 شرطي اكثر من العام الماضي.</p>
<p>اما اثونيسيا التي شهدت اعتداءات نفذتها «الجماعة الاسلامية»، الحليفة الاقليمية لتنظيم «القاعدة» الارهابي، فجندت لرأس السنة ثلثي قوات الشرطة لديها وعددهم 250 الفا، وامر انها لم تتلق اي تهديد محدد.</p>
<p>غير ان احتفالات رأس السنة لن تجري كما كان متوقعا في كل انحاء العالم، فقد الغيت الاحتفالات في اسلام اباد تحسبا لاتخاذ قمة اقليمية تشارك فيها الهند وباكستان اهتمامي استهدافها الرئيس برويز مشرف.</p>
</div1>
- <div1 n="2" d="NEWS-05">
```

Figure 6.2: An Example of **CCA** in XML Format

All articles in **CCA** are downloaded, with the size around **10.3MB**. In order to present this corpus in a suitable form to be useful in this study, the **annotated corpus** should be generated. The steps of generating such corpus are presented as follows:

1. The **CCA** is cleaned from all XML mark-up tags, numbers, punctuations, and special symbols (see Section 5.2.1 of Chapter 5).

2. The **CCA** is tokenised (see Section 5.2.2 of Chapter 5) into a set of Ara-

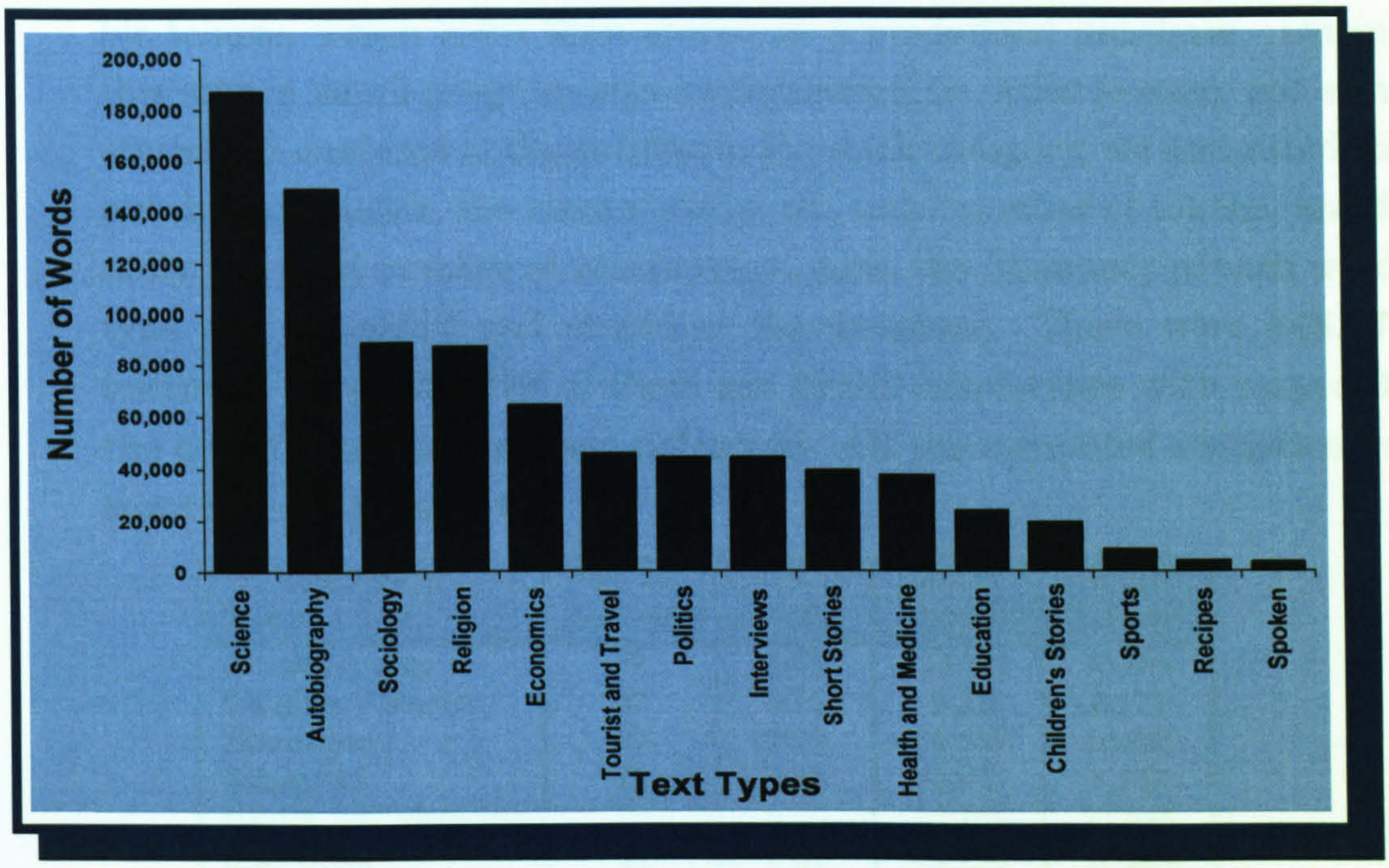


Figure 6.3: Number of Tokens in Each Category for **CCA**

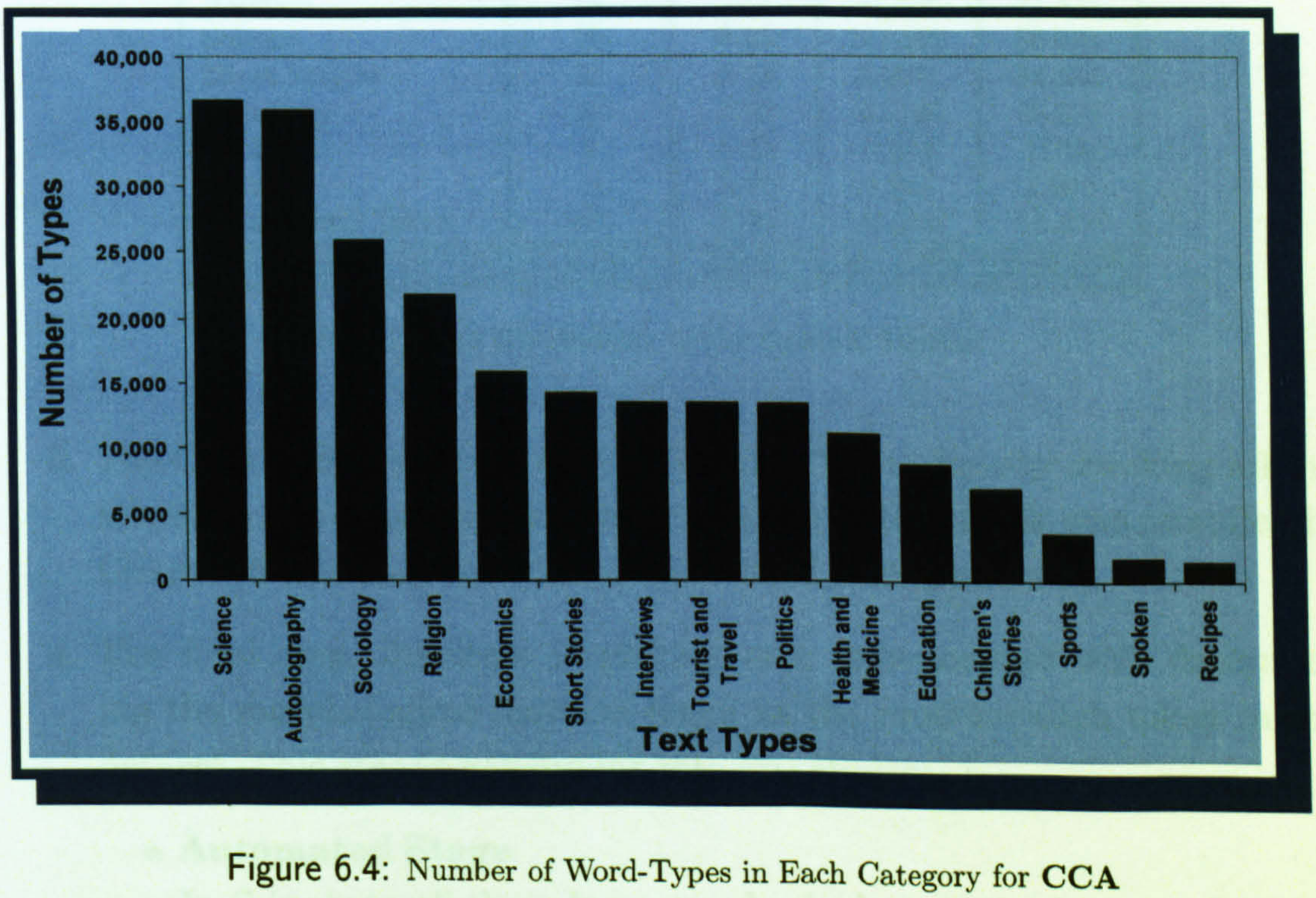


Figure 6.4: Number of Word-Types in Each Category for **CCA**

bic tokens, which were then stored in a predefined database. During this step, a small program was implemented, in order to carry out some statistical analyses of these tokens. For each category, we computed the number of articles, the memory size, the total number of tokens, and finally, the total number of word-types[4]. Also, the frequency of each word-type was computed and stored in the database. There were **848,779** tokens in **CCA**, **225,186** of them are Arabic word-types with respect to the categories that they are belong to. All the computed statistics are summarised in Table 6.1.

Category Type	No. of Articles	Size in MB	No. of Tokens	Word Tokens
Autobiography	72	1.65	149,531	35,884
Children's Stories	27	0.26	19,322	6,972
Economics	29	0.78	64,505	16,033
Education	10	0.41	23,895	8,846
Health and Medicine	32	0.45	37,165	11,245
Interviews	23	0.61	44,165	13,584
Politics	10	0.52	44,307	13,494
Recipes	9	0.07	4,581	1,500
Religion	19	1.17	87,836	21,833
Science	70	2.23	187,222	36,698
Short Stories	31	0.43	39,071	14,284
Sociology	30	0.99	89,196	25,877
Spoken	5	0.05	3,632	1,778
Sports	4	0.10	8,641	3,602
Tourist and Travel	60	0.60	45,710	13,556
Total	**431**	**10.32**	**848,779**	**225,186**

Table 6.1: The Distribution of the Original Dataset

3. The **CCA** is cleaned from the stopwords. This is done by checking all the tokens in the **dataset** against the stopwords list which was compiled in Chapter 5 (the stopwords list is presented in Appendix C).

4. The final step, the most important one, is concerned with determining the morphological features (such as the root) for each token in the dataset. This step is subdivided into two stages:

   - **Automated Stage**

     In this stage all the tokens are checked against a lexicon of Arabic

[4]Word-types mean identical or distinct tokens.

artificial data (AAD). The **AAD** is generated by the author by interdigitising all the roots in the **common dictionary**[5] with all the compiled **lexical patterns**[6]. The generated **AAD** has 23,887,767 tokens (12,619 roots x 1,893 patterns). Of course, many of these tokens are invalid Arabic tokens and for this reason we called them as artificial data.

All the tokens in the **CCA** are checked against the **ADD**. If there is a match, then the root and the pattern corresponding to this token are retrieved. Otherwise, the given token is processed in the manual stage.

- **Manual Stage**
  In this stage, all the tokens that were not processed in the automated stage are rooted[7] by hand. Every assigned root is checked against the **common-dictionary** that was created in Chapter 4 in order to check if the assigned root is a valid root. To guarantee more adequate level of accuracy, an Arabic linguist[8] was consulted during this step.

The output of all the previous step results is the **annotated CCA** with **66,344** word-types, in which the morphological information about each token is presented. Figure 6.5 shows a snapshot of the **annotated CCA** for the Arabic tokens with length **10**.

The **annotated CCA** as shown in Figure 6.5 consists of eight columns, the description of these columns are as follows:

- The first column, for the Arabic token.
- The second column, for the frequency of each token in the **CCA**.

[5]There are 12,619 roots in the common dictionary (see Chapter 4).
[6]There are 1,893 compiled patterns (see Chapter 4).
[7]Assign a root and a pattern to the token.
[8]Fahed Ashour, email: fahed.ashour@gsaa.uni-halle.de. Martin Luther University, Halle, Germany.

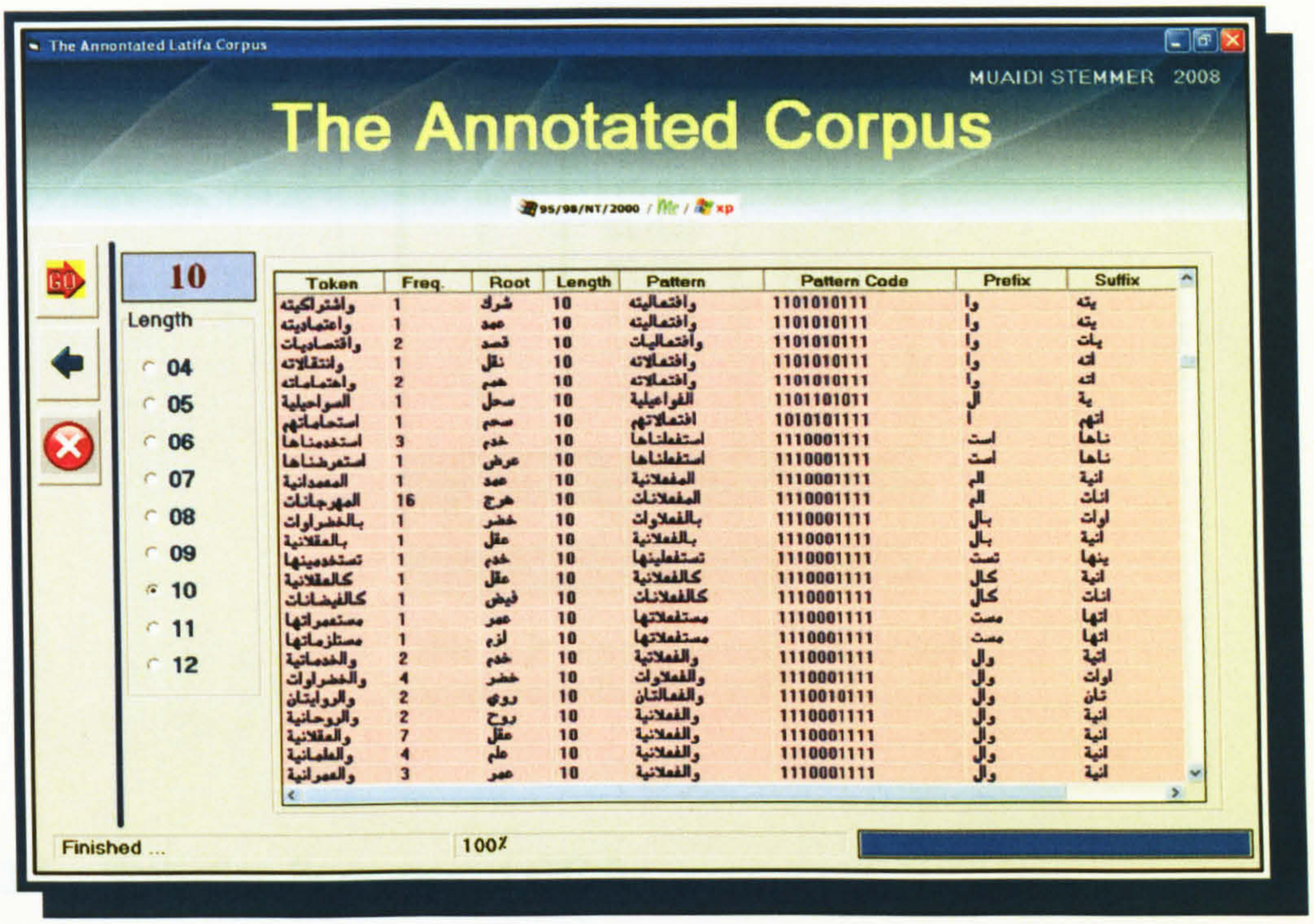


Figure 6.5: A Snapshot of the Annotated **CCA**

- The third column, for the triliteral root of the token.
- The fourth column, for the length of the token.
- The fifth column, for the morphological pattern corresponding to the token.
- The sixth column, for the binary pattern code.
- The seventh column, for the prefix string of the token.
- The eighth column, for the suffix string of the token.

All the tokens in the **annotated CCA** are divided into nine sub-groups according to their lengths. This is done to follow the presented **MPBNNs** approach (see Section 6.2 of this chapter). The number of tokens, and the number of word-types in each group are computed. Table 6.2 tabulated these computations.

Token Length	No. of Tokens	Word Types
4	152,408	13,565
5	156,670	24,030
6	125,540	25,415
7	94,924	19,600
8	43,795	11,397
9	17,621	5,078
10	6,240	1,910
11	2,097	710
12	621	282
Total	**596,916**	**101,987**
Others	251,863	3,911
Total	**848,779**	**105,898**

Table 6.2: Analysis of the Annotated Corpus According to the Word Lengths

6.3.1 Statistics Summary of CCA

In this subsection, all the statistic analysis results of the **CCA** and the **annotated CCA** are presented. Figure 6.6 draws the map of these results. Every object in this figure has three information rows. The top row for the object title itself, the second one for the number of tokens (including redundant tokens), while the last one for the number of word-types.

The following items highlight the results in Figure 6.6.

1. The **CCA** consists of **848,779** tokens (including redundant tokens).
2. The minimum token length in this corpus is **1**. In this study, the tokens have three letters or less are ignored. The reason behind this comes from the assumption that says: "to extract the trilieral (3 consonants) root for a given token, then this token should have an extra affix letter".
3. The maximum token length in this corpus is **25**, while the maximum valid Arabic tokens length is **12**. This has coincided with the pattern analysis that was presented in Chapter 4. The tokens which have a length greater than **12** are borrowed tokens or scientific English tokens

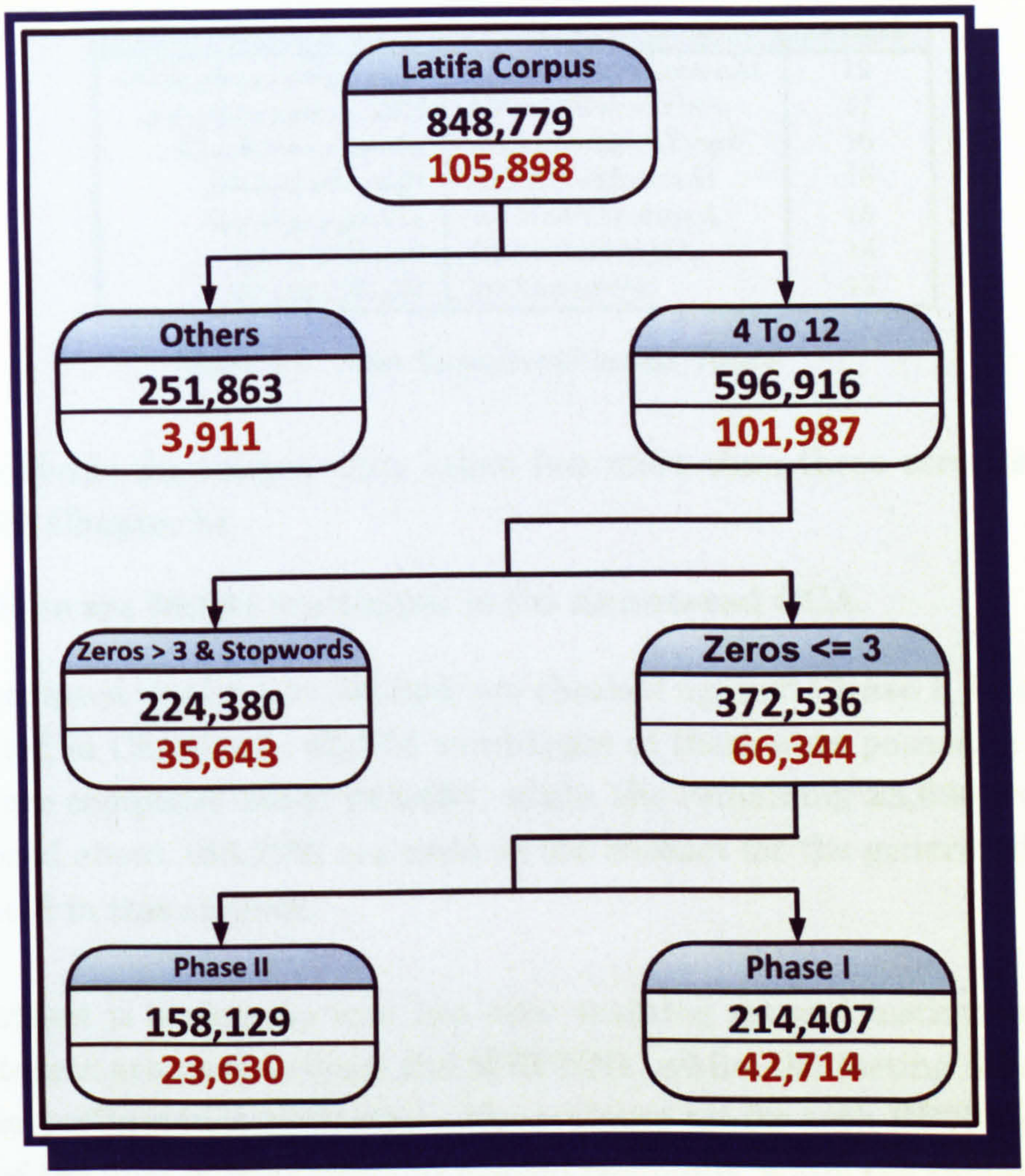


Figure 6.6: Statistical Summary of **CCA**

written in Arabic scripts. Table 6.3 shows some examples of these invalid tokens.

4. The total number of tokens which have a length greater than or equal **4** and less than or equal **12** is **596,916**, about **101,987** of them are word-types.

5. Only **372,536** from **596,916** are annotated tokens, i.e. tokens have roots. The reason for the gap is that some Arabic tokens are mainly borrowed, and some of them are stopwords. The number of zeros test is used here

Token	Transliteration	Length
الكلوروفلوروكربونات	Alklwrwflwrwkrbwnat	19
الكلوروفلوروكربون	Alklwrwflwrwkrbwn	17
والكهرومغناطيسية	wAlkhrwmgnATysyt	16
الإنتركونتيننتال	AlĀntrkwntynntAl	16
والأنثروبولوجيا	wAlĀnt̂rwbwlwjyA	15
البيوتكنولوجيا	Albywtknwlwjya	14
للترانزستورات	lltrAnzstwrAt	13

Table 6.3: Some Examples of Invalid Tokens

to divide the tokens. Any token has more than three zeros is ignored (see Chapter 5).

6. There are **66,344** word-types in the **annotated CCA**.

The annotated word-types (**66,344**) are checked against **Phase I**. **Phase I** was presented in Chapter 5. **42,714** word-types of them were passed in **Phase I** which are composed about **214,407**, while, the remaining **23,630** word-types (composed about **158,129**) are used as the dataset for the generated **BPNNs** presented in this chapter.

The dataset is broken up into two sets: training set and testing set. Training datasets are used to train the **MBPNNs**, while, the testing data sets are used for performance evaluation. The training set for each **BPNN** is roughly made of the **75%** of the examples chosen randomly, while the remaining **25%** constitutes the testing set. The exact number of samples for each **BPNN** is tabulated in Table 6.4.

6.4 Data Preparation

6.4.1 Input and Output Coding

The input coding is based on the weight values that were presented in Section 5.2.5 of Chapter 5. In this coding, every Arabic letter is assigned a decimal value extracted from the frequency of a letter to appear as an affix letter.

BPNN	Word Types	Training Tokens	Testing Tokens
$BPNN_4$	4,331	3,248	1,083
$BPNN_5$	7,719	5,789	1,930
$BPNN_6$	5,229	3,922	1,307
$BPNN_7$	2,902	2,177	725
$BPNN_8$	1,434	1,076	358
$BPNN_9$	1,579	1,184	395
$BPNN_{10}$	378	284	94
$BPNN_{11}$	58	44	14
Total	**23,630**	**17,724**	**5,906**

Table 6.4: The Size of the Training Set and Testing Set for all the Developed BPNNs

For the output coding, the main aim of this study is to extract the root of a given input word. The root-letters are a subset of the input word letters. For example, in Arabic tokens with a length equal to 4, three of these four letters are root-letters, and the remaining extra letter is an affix letter. The positions of the root-letters are in one of the following positions: [1,2,3], [1,2,4], [1,3,4], [2,3,4].

As this example suggests, in all Arabic tokens with a length = 4, there are 4 different classes for the root-letter positions. To encode these classes, we used **1-of-n** output encoding. The **1-of-n** encoding represents the output as a vector. The length of this vector is equal to the number of discrete classes allowed for the variable, where every element in the code vector is set to **0**, with the exception of the one corresponding to the right class, which is set to **1**. For example, Table 6.5 shows how to encode the target classes for the $BPNN_4$ which represents the ANN for Arabic tokens with length **4**.

Word's Length	Root Letter Positions	Target Vector	Class
4	1, 2, 3	0001	1
4	1, 2, 4	0010	2
4	1, 3, 4	0100	3
4	2, 3, 4	1000	4

Table 6.5: **Target Encoding for $BPNN_4$**

There are two motivations for choosing the **1-of-n** output encoding. First,

it is simple, easy to use, and the ANN can easily be learning to discriminate between the various values. Second, it provides more degrees of freedom to the network for representing the target vector.

6.4.2 Data Scaling

Using the raw data as input may lead the ANN architecture to saturation (i.e. one input or more may become dominant). So, it is necessary to normalise the input data. The Scaling is the process of bringing the input data columns at the same interval. Thus, scaling is needed for an effective training and it improves ANNs performance [76]. Inputs to ANNs are commonly normalised between −1 and 1 using the following equation [1]:

$$xn = \frac{2 * (x - minx)}{(maxx - minx)} - 1 \tag{6.1}$$

Where:

xn : is the standardised input.
x : is the current input.
$minx$: is the lowest input in all samples.
$maxx$: is the highest input in all samples.

Figure 6.7 shows the data scaling for the Arabic word [والمصانع] (wAlmSAn', "and the factories"). This word has an input vector [18, 20, 14, 15, 0, 20, 16, 0]. The data scaling was applied assuming that $minx = 0$ and $maxx = 20$ for all vector elements.

6.5 Neural Network Topologies

As mentioned earlier, **eight** BPNNs are developed , each of them is for a specific word length. We labelled them from $BPNN_4$ to $BPNN_{11}$. The $BPNN_4$ for Arabic tokens with length = 4, while the $BPNN_{11}$ for Arabic tokens with length = 11 (See Figure 6.1 Page 154). All the developed BPNNs have two

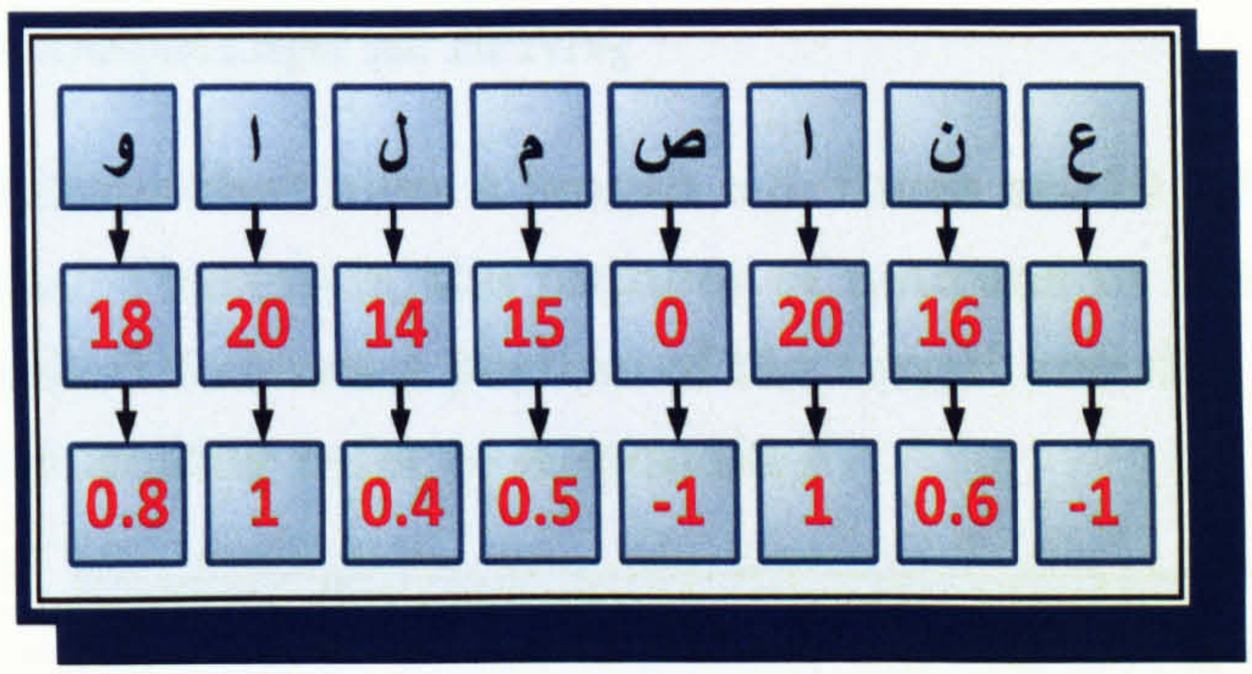


Figure 6.7: Data Scaling for the Arabic Word [والمصانع](*wAlmSAn'*)

layers: output and hidden layers, in addition to the input layer. Every BPNN was presented with a word's weight vector at its input layer, and was trained to produce the corresponding root vector class at its output layer.

6.5.1 Size of Input Layer

The number of input nodes in each BPNNs is equal to the length of the corresponding input Arabic word, which is between 4 and 11. So, the number of input nodes in $BPNN_4$ is 4, in $BPNN_5$ is 5, while in $BANN_{11}$ is 11.

6.5.2 Size of Output Layer

Normally, the number of output nodes is the same as the number of different classes that a ANN is trained to recognise. So, the number of output nodes in the developed BPNNs varies from one network to another depending on the number of root position classes in each BPNN.

We follow the same procedure for output encoding that was presented in Section 6.4.1. The following subsections highlight some of the interesting things we have observed while studying and analysing the root and the affix positions in the Arabic tokens.

6.5.2.1 Size of the Output Layer for $BPNN_5$

For the Arabic tokens that have a length = 5, there are 10 different classes for the root positions. Therefore, the number of neurons in the output layer for $BPNN_5$ is 10. Table 6.6 shows a portion of the target encoded for $BPNN_5$. For a complete target encoding refer to Appendix F.

Word's Length	Root Letter Positions	Target Vector	Class
5	1, 2, 3	0000000001	1
5	1, 2, 4	0000000010	2
5	1, 2, 5	0000000100	3
.	.	.	.
.	.	.	.
5	3, 4, 5	1000000000	10

Table 6.6: A Portion of the Target Encoded for $BPNN_5$

6.5.2.2 Size of the Output Layer for $BPNN_6$

For the Arabic tokens that have a length = 6, theoretically there are 20 different choices to choose 3 positions (for root letters) from 6 positions (for word's length). But during our analysis we found that only 14 from these 20 choices have root letters. So, the length of the target vector decreased from 20 bits to 14 bits. Therefore, the number of neurons in the output layer for $BPNN_6$ is 14. Table 6.7 shows a portion of the target encoded for $BPNN_6$. For a complete target encoding refer to Appendix F.

Word's Length	Root Letter Positions	Target Vector	Class
6	1, 2, 3	00000000000001	1
6	1, 2, 4	00000000000010	2
6	1, 3, 4	00000000000100	3
.	.	.	.
.	.	.	.
6	4, 5, 6	10000000000000	14

Table 6.7: A Portion of the Target Encoded for $BPNN_6$

6.5.2.3 Size of the Output Layer for $BPNN_7$

For the Arabic tokens that have a length = 7, theoretically there are 35 different choices to choose 3 positions (for root letters) from 7 positions (for word's length). But during our analysis we found that only 16 from these 35 choices have root letters. So, the length of the target vector decreased from 35 bits to 16 bits. Therefore, the number of neurons in the output layer for $BPNN_7$ is 16. Table 6.8 shows a portion of the target encoded for $BPNN_7$. For a complete target encoding refer to Appendix E.

Word's Length	Root Letter Positions	Target Vector	Class
7	1, 2, 3	0000000000000001	1
7	1, 2, 4	0000000000000010	2
7	1, 3, 4	0000000000000100	3
.	.	.	.
.	.	.	.
7	5, 6, 7	1000000000000000	16

Table 6.8: A Portion of the Target Encoded for $BPNN_7$

6.5.2.4 Size of the Output Layer for $BPNN_8$

For the Arabic tokens that have a length = 8, theoretically there are 56 different choices to choose 3 positions (for root letters) from 8 positions (for word's length). But during our analysis we found that only 19 from these 56 choices have root letters. So, the length of the target vector decreased from 56 bits to 19 bits. Therefore, the number of neurons in the output layer for $BPNN_8$ is 19. Table 6.9 shows a portion of the target encoded for $BPNN_8$. For a complete target encoding refer to Appendix E.

6.5.2.5 Size of the Output Layer for $BPNN_9$

For the Arabic tokens that have a length = 9, theoretically there are 84 different choices to choose 3 positions (for root letters) from 9 positions (for word's length). But during our analysis we found that only 30 from these 84 choices have root letters. So, the length of the target vector decreased from 84 bits to

Word's Length	Root Letter Positions	Target Vector	Class
8	1, 2, 3	00000000000000000001	1
8	1, 2, 4	00000000000000000010	2
8	2, 3, 4	00000000000000000100	3
.	.	.	.
.	.	.	.
8	5, 7, 8	10000000000000000000	19

Table 6.9: A Portion of the Target Encoded for $BPNN_8$

30 bits. Therefore, the number of neurons in the output layer for $BPNN_9$ is 30. Table 6.10 shows a portion of the target encoded for $BPNN_9$. For a complete target encoding refer to Appendix E.

Word's Length	Root Letter Positions	Target Vector	Class
9	1, 2, 3	000000000000000000000000000001	1
9	1, 2, 4	000000000000000000000000000010	2
9	1, 2, 5	000000000000000000000000000100	3
.	.	.	.
.	.	.	.
9	7, 8, 9	100000000000000000000000000000	30

Table 6.10: A Portion of the Target Encoded for $BPNN_9$

6.5.2.6 Size of the Output Layer for $BPNN_{10}$

For the Arabic tokens that have a length = 10, theoretically there are 120 different choices to choose 3 positions (for root letters) from 10 positions (for word's length). But during our analysis we found that only 26 from these 120 choices have root letters. So, the length of the target vector decreased from 120 bits to 26 bits. Therefore, the number of neurons in the output layer for $BPNN_{10}$ is 26. Table 6.11 shows a portion of the target encoded for $BPNN_{10}$. For a complete target encoding refer to Appendix E.

6.5.2.7 Size of the Output Layer for $BPNN_{11}$

For the Arabic tokens that have a length = 11, theoretically there are 165 different choices to choose 3 positions (for root letters) from 11 positions (for

Word's Length	Root Letter Positions	Target Vector	Class
10	1, 7, 8	00000000000000000000000001	1
10	2, 3, 4	00000000000000000000000010	2
10	2, 3, 5	00000000000000000000000100	3
.	.	.	.
.	.	.	.
10	7, 8, 10	10000000000000000000000000	26

Table 6.11: A Portion of the Target Encoded for $BPNN_{10}$

word's length). But during our analysis we found that only 17 from these 165 choices have root letters. So, the length of the target vector decreased from 165 bits to 17 bits. Therefore, the number of neurons in the output layer for $BPNN_{11}$ is 17. Table 6.12 shows a portion of the target encoded for $BPNN_{11}$. For a complete target encoding refer to Appendix E.

Word's Length	Root Letter Positions	Target Vector	Class
11	2, 4, 6	00000000000000001	1
11	3, 4, 6	00000000000000010	2
11	3, 5, 6	00000000000000100	3
.	.	.	.
.	.	.	.
11	8, 9, 10	10000000000000000	17

Table 6.12: A Portion of the Target Encoded for $BPNN_{11}$

The complete list of the target encoding for all *BPNNs* is shown in Appendix E. Table 6.13 shows the number of neurons in the output layer for each of the developed BPNNs.

6.5.3 BPNN Parameter Optimisation

Depending on the type of the learning algorithm and BPNN used, there are some parameters that must be set, in order to control the training process. The next two subsections deal with the most frequently encountered parameters: size of the hidden layer and the learning rate.

BPNN	Output Neurons
$BPNN_4$	4
$BPNN_5$	10
$BPNN_6$	14
$BPNN_7$	16
$BPNN_8$	19
$BPNN_9$	30
$BPNN_{10}$	26
$BPNN_{11}$	17

Table 6.13: Number of Neurons in the Output Layer for each BPNNs

6.5.3.1 Optimal Size of the Hidden Layer

The first parameter to be determined is the number of hidden neurons in the hidden layer. The ANN architecture can have more than one hidden layer. But, in several research studies, one hidden layer is sufficient for most types of problems [116]. In all the developed BPNNs there is only one hidden layer. The number of hidden neurons in this hidden layer affects the generalisation performance [123]. As the number of neurons is increased or decreased, BPNNs can suffer from either underfitting or overfitting [76].

UNDERFITTING When the number of hidden layer neurons is low, the network cannot approximate the target values close to the output values. The network generalises too much. This is called underfitting.

OVERFITTING When the hidden layer neurons are too many, the network starts memorising instead of generalising. This is called overfitting. Memorising occurs because the extra connections generated by the higher number of neurons are able to distinguish between each input.

Since there is no rule for determining the optimal number of hidden neurons, we used a series of trial and error experiments in order to determine the ideal number of these neurons. For each developed BPNN, we tested 40 different values for the number of neurons in the hidden layer. In each test, we started

by an initial guess value. This value is found in many ANN literature and is advised to be taken as an initial value [76]. The initial value is computed as shown in Equation 6.2.

$$hidden\ layer\ size = \sqrt{input\ layer\ size\ *\ output\ layer\ size} \tag{6.2}$$

For example, for $BPNN_5$, the initial guess value is $7 = \sqrt{5 * 10}$, where **5** is the size of the input layer and **10** is the size of the output layer for $BPNN_5$. Table 6.14 shows all the initial guess values and the final values for the size of hidden layers in all of the developed *BPNNs*.

BPNN	Input Layer Size	Output Layer Size	Initial Value	Final Value
$BPNN_4$	4	4	4	44
$BPNN_5$	5	10	7	47
$BPNN_6$	6	14	9	49
$BPNN_7$	7	16	11	51
$BPNN_8$	8	19	12	52
$BPNN_9$	9	30	16	56
$BPNN_{10}$	10	26	16	56
$BPNN_{11}$	11	17	14	54

Table 6.14: The Initial & Final Values for Determining Hidden Layer Size

In each setting, **5** independent runs are conducted, where different random initial weights are used in each run. This process counts up to 1600 experiments in total (8 different BPNNs x 40 neurons x 5 runs). During each experiment every developed BPNN was allowed to train as far as 10,000 epochs, although in all cases training stopped before reaching this number due to the low error in the training set. The primary metric for determining network performance is the mean squared error *MSE*, which is the square of the average differences between each actual network output and the desired output.

Table 6.15 shows as an example all the experiments used to find the optimal number of hidden neurons for the $BPNN_4$, while Appendix F shows the performance of each developed BPNN for the training set when a different number of neurons is applied in BPNN's hidden layer.

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	AVG. MSE
4	0.1145561	0.1148164	0.1216486	0.1143701	0.1193599	0.1169502
5	0.1113855	0.1247833	0.1115374	0.1137931	0.1113290	0.1145657
6	0.1030022	0.1113334	0.1090625	0.1101775	0.1127836	0.1092718
7	0.1067190	0.1091654	0.1066213	0.1048473	0.1039767	0.1062659
8	0.1039348	0.0975601	0.0945494	0.1044778	0.1063969	0.1013838
9	0.0996528	0.0937576	0.1016259	0.1049016	0.1037443	0.1007364
10	0.0959925	0.0977803	0.0936786	0.0933737	0.0942179	0.0950086
11	0.0953250	0.1118523	0.0907649	0.0948221	0.0951151	0.0975759
12	0.0926278	0.0909209	0.0902183	0.0924597	0.0947102	0.0921874
13	0.0899816	0.0904081	0.0924370	0.0879367	0.0934493	0.0908425
14	0.0900568	0.0908192	0.0941017	0.0946975	0.1017401	0.0942831
15	0.0961057	0.0915545	0.0887634	0.0909342	0.0894662	0.0913648
16	0.0914085	0.0879125	0.0909122	0.0899075	0.0893352	0.0898952
17	0.0877792	0.0916783	0.0922123	0.0970942	0.0882738	0.0914076
18	0.0877161	0.0909515	0.0903733	0.0991982	0.0858846	0.0908247
19	0.0989957	0.0917288	0.0903173	0.0882167	0.0877338	0.0913985
20	0.0888636	0.0893121	0.0918632	0.0868360	0.0925849	0.0898920
21	0.0907992	0.0943698	0.0919475	0.0892845	0.0932196	0.0919241
22	0.0877585	0.0899058	0.0903428	0.0912020	0.0857219	0.0889862
23	0.0890485	0.0897936	0.0876362	0.0909045	0.0879124	0.0890590
24	0.0877001	0.0881523	0.0842669	0.0848657	0.0867751	0.0863520
25	0.0872173	0.0841889	0.0889574	0.0876747	0.0876458	0.0871368
26	0.0957700	0.0883402	0.0883138	0.0896394	0.0880221	0.0900171
27	0.0878897	0.0866275	0.0870780	0.0860570	0.0866096	0.0868524
28	0.0870180	0.0872560	0.0884963	0.0880741	0.0892668	0.0880222
29	0.0865603	0.0888585	0.0891339	0.0818730	0.0856603	0.0864172
30	0.0836018	0.0860587	0.0877835	0.0872850	0.0867758	0.0863010
31	0.0912445	0.0854871	0.0814105	0.0858626	0.0822439	0.0852497
32	0.0875797	0.0873709	0.0876831	0.0848809	0.0878095	0.0870648
33	0.0864475	0.0902359	0.0834274	0.0862727	0.0851515	0.0863070
34	0.0887363	0.0893111	0.0858644	0.0966402	0.0851859	0.0891476
35	0.0874258	0.0830726	0.0900899	0.0874742	0.0870544	0.0870234
36	0.0916865	0.0862568	0.0942285	0.0842586	0.0863117	0.0885484
37	0.0878176	0.0828580	0.0864355	0.0847201	0.0861602	0.0855983
38	0.0855029	0.0846637	0.0848990	0.0840298	0.0867500	0.0851691
39	0.0853720	0.0868711	0.0903840	0.0836312	0.0855300	0.0863577
40	0.0835249	0.0814149	0.0863844	0.0830406	0.0832820	0.0835294
41	0.0912577	0.0887260	0.0846866	0.0856289	0.0871020	0.0874802
42	0.0873604	0.0857965	0.0849970	0.0839161	0.0849319	0.0854004
43	0.0871074	0.0884211	0.0856455	0.0897413	0.0885783	0.0878987

Table 6.15: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_4$

Finally, Table 6.16 summarises the best number of hidden neurons which are found in each developed BPNN in terms of the performance of the training set.

BPNN	Optimal Number of Hidden Neurons
$BPNN_4$	40
$BPNN_5$	44
$BPNN_6$	43
$BPNN_7$	49
$BPNN_8$	48
$BPNN_9$	53
$BPNN_{10}$	46
$BPNN_{11}$	29

Table 6.16: The Optimal Number of Neurons in the Hidden Layer for each BPNNs

6.5.3.2 Optimal Value of the Learning Rate

The other important training parameter is the optimal value of learning rate. Almost all ANN models have a learning rate parameter associated with them. According to Attoh-Okine [43] the learning rate is "a parameter that determines the size of the weights adjustment each time the weights are changed during training".

Choosing an adequate value for the learning rate parameter has a significant effect over ANN performance. Usually, choosing a small value for the learning rate, the BPNN will need a large number of training epochs. As a consequence it would take a long learning time. To avoid this situation, one may consider to choose a relatively large learning rate. In this case, the process of learning is faster, but unfortunately, this may lead to over adjustment of weights which may occur in the error surface regions with extreme steepness.

The BPNN algorithm with a fixed learning rate would not be efficient [84]. Therefore, an efficient learning algorithm should be able to dynamically vary its learning rate in accordance with changes of the gradient values. This way of learning is the so-called adaptive learning rate. An adaptive learning rate improves the speed of training for the BPNNs [65]. The way for using the adaptive learning rate during training the BPNN is as follows, these steps

are adopted from [2].

1. First the initial network output and error are calculated.
2. At each epoch new weights are computed using the current learning rate.
3. The new output and error are calculated.
4. If the new error exceeds the old error by more than a predefined ratio of 1.04, the new weights and errors are discarded. In addition, the learning rate is decreased by 0.7.
5. Otherwise, the new weights, and error are stored. If the new error is less than the old error, the learning rate is increased by 1.05.

Having all the above, in this study, we used the adaptive learning rate instead of the fixed learning rate. The values generated for the adaptive learning rates are presented in Table 6.17. For each BPNN the initial and final values are illustrated.

BPNN	Initial Value	Final Value
$BPNN_4$	0.050	119.38
$BPNN_5$	0.050	1567.98
$BPNN_6$	0.050	1889.76
$BPNN_7$	0.050	1675.45
$BPNN_8$	0.050	1789.08
$BPNN_9$	0.050	2458.03
$BPNN_{10}$	0.050	1337.12
$BPNN_{11}$	0.050	1615.05

Table 6.17: Adaptive Learning Rates for all BPNNs

6.6 Training The Neural Networks

All the developed BPNNs are trained prior to being deployed an a test dataset for root extraction. As mentioned earlier, all the BPNNs are trained by the backpropagation learning rule based on a supervised learning paradigm. In

order to train the BPNN a set of training word examples and a set of their correct root classes are provided, each training example (T_i) is an input-output pair, as shown in Equation 6.3.

$$T_i = (X_i, O_i) \tag{6.3}$$

where X_i presents the scaled weight values of the word letters in each example, and O_i presents the correct root-letter positions class. Each of BPNNs represents a group of the data and is trained separately. Every BPNN has its own topology, Table 6.18 summarises the topologies of all the developed BPNNs. In this table, the number of neurons of a specific network is indicated by the triple [I,H,O] where **I** is the number of neurons in the input layer, **H** the same for the hidden layer, and **O** for the output layer.

BPNN	[I,H,O]
$BPNN_4$	[04,40,04]
$BPNN_5$	[05,44,10]
$BPNN_6$	[06,43,14]
$BPNN_7$	[07,49,16]
$BPNN_8$	[08,48,19]
$BPNN_9$	[09,53,30]
$BPNN_{10}$	[10,46,26]
$BPNN_{11}$	[11,29,17]

Table 6.18: The Topologies of the Developed BPNNs

There are some fixed parameters used during the training processes, which are summarised in Table 6.19. The evaluation and the performance of the developed BPNNs are discussed at length in Chapter 7.

Activation Function	:	The Log-Sigmoid activation function for the output and hidden layers.
Performance Function	:	MSE.
Learning Rate	:	Adaptive Learning Rate.
Learning Algorithm	:	Backpropagation.
Maximum Epochs	:	10,000.
Number of hidden layers	:	1.

Table 6.19: The Summary of the Parameters that are Used in the Training Processes for all the Developed BPNNs

6.7 Root Extraction As A Trained Classification Problem

In ANN the task of a classification problem is to determine the correct class for a particular object according to its characteristics or features on a particular domain. For example, given a set of a mathematical shapes, it is often desirable to divide the shapes into classes such as squares, rectangles and triangles . In this case, the features based on which the shapes are separated into classes may be the number of ribs, the value of angles and the relations between the rib lengths.

In this chapter, the extraction of Arabic word roots is formulated as a classification problem and the ANN as a classifier tool. The term object here is refer to the Arabic word itself, while the term class is refer to the combination of the root-letter positions. The general model of the ANN classifier used in this research can be viewed as shown in Figure 6.8.

In the classifier model shown in Figure 6.8, the vector $<X_1, X_2, \ldots, X_n>$ presents the features of the input Arabic word which are here the scaled weight values of the letters that are composed this word. The index (n) in the input vector is for the word length.

The trained ANN classifier then makes its decision by producing an m-dimensional output vector $<O_1, O_2, \ldots, O_m>$ presents the correct root-letters class. The index (m) in the class vector is for the number of classes that is

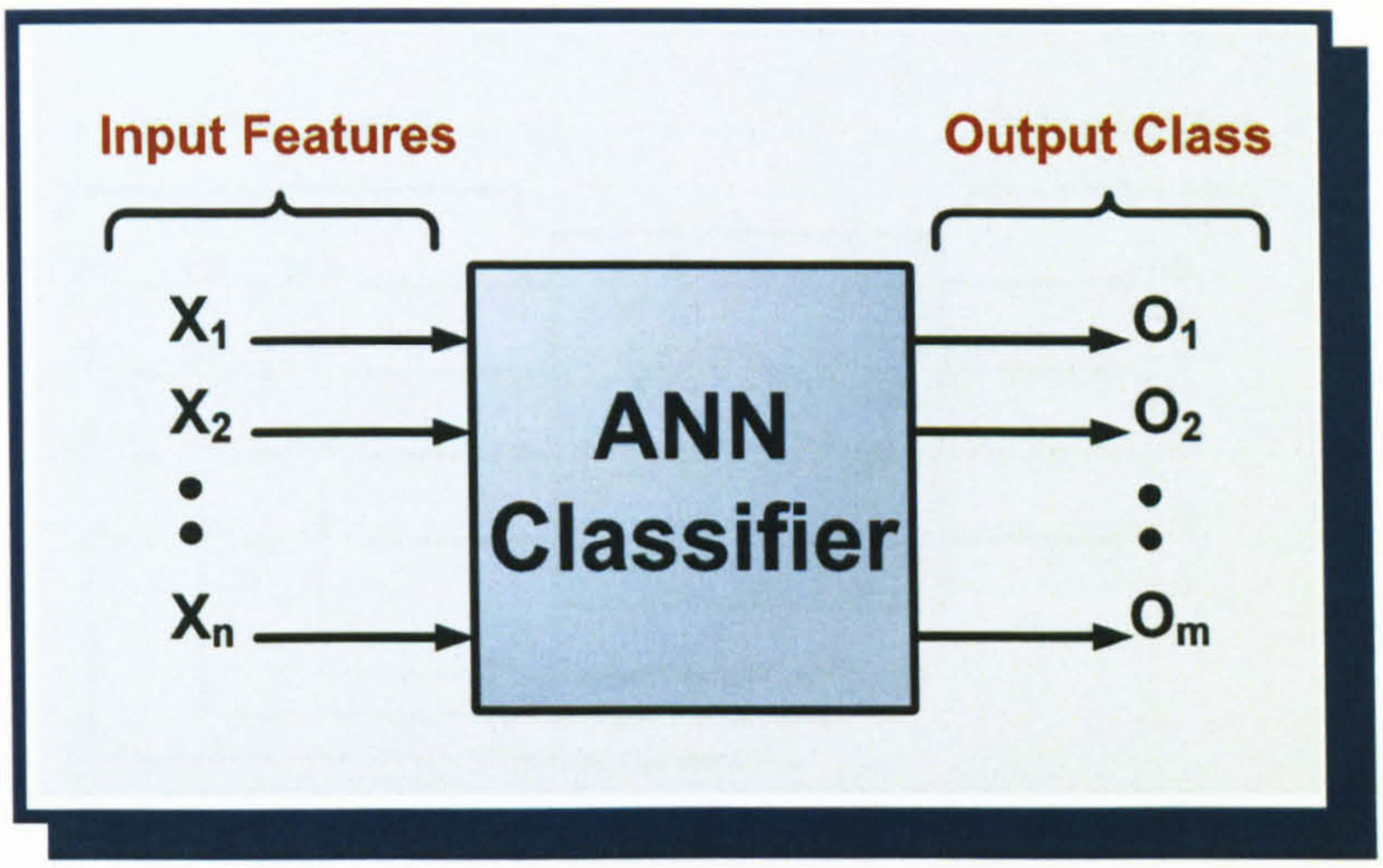


Figure 6.8: The General Model of the ANN Classifier

already predefined for this ANN. More formally, the ANN classifier can seen as a function that maps the n-dimensional input vector features to the m-dimensional output vector class, which is clearly shown in Equation 6.4.

$$f(<X_1, X_2, \ldots, X_n>) = <O_1, O_2, \ldots, O_m> \tag{6.4}$$

The following example clears how the ANN classifier tries to extract the correct root for a given Arabic word.

Example

Assume the tested word is [ماهر](mAhr,"proficient"), this word has 4 letters, which are they, L ={[م](m), [ا](A), [ه](h), [ر](r)}. The corresponding weight values are W = <15, 20, 13, 0>. The scaled weight values are W' = <0.5, 1, 0.3, -1> (see Section 6.4.2 Page 166 of this chapter). The scaled weight values are fed to the input layer of the $BPNN_4$ classifier as shown in Figure 6.9.

The trained $BPNN_4$ classifier in Figure 6.9 is produced the output vector O = <0100>. According to the target values that are illustrated in Appendix E, this output vector is for the class that presents the root-letter positions as [1,3,4], which means that the root-letters are the first, the third, and the

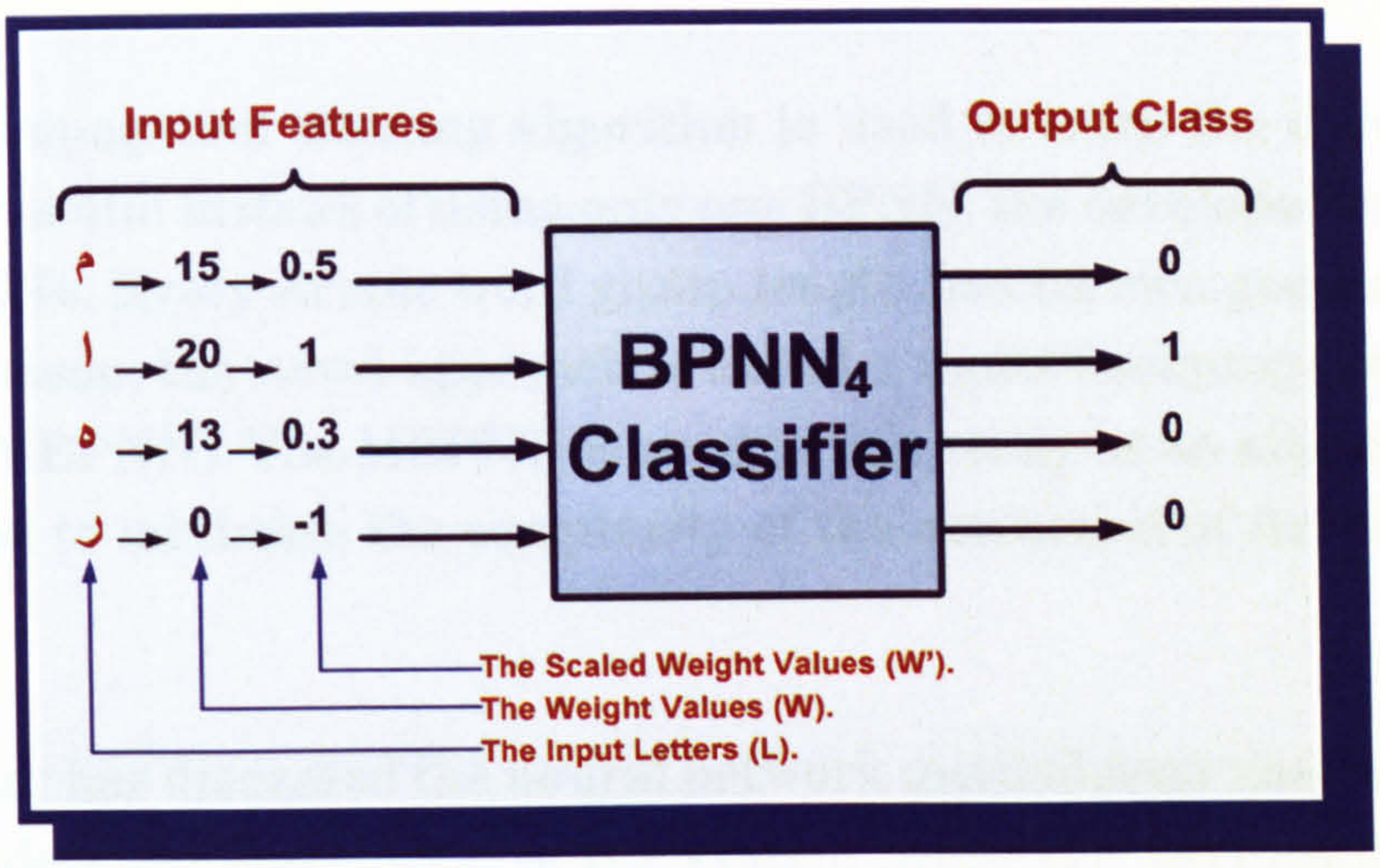


Figure 6.9: An Example of Extraction the Root Using BPNN Classifier

fourth letters in the current example. These letters are {[م](m), [ه](h), [ر](r)}. By combining these letters, the root [مهر](mhr) is extracted which is the correct root.

All the training experiments presented in this chapter were trained using the **MATLAB** software version 6.5. A program in **MATLAB** language is written by the author for this purpose. The **MATLAB** environment is just used for the training purpose, the other programming functions used in this thesis are programmed by the author using **Visual Basic** version 6.0. A full GUI system is implemented for the purpose of this research. All the snapshot figures presented in this thesis are extracted from this system.

6.8 Summary

Artificial neural networks are massive parallel computing machines inspired by nervous systems. This chapter presents the second phase of the developed solution to the extraction of the roots of Arabic words problem which is based on the artificial neural networks. The novel developed solution deals with the root extraction problem as a classification problem and the artificial neural network as a classifier tool created by supervised learning.

The backpropagation training algorithm is used to learn the developed neural networks and instead of using only one BPNN, the developed solution uses eight BPNNs. Every Arabic word group length has its own generated BPNN. For this reason, the novel approach is called a multi-backpropagation neural network (MBPNN). The MBPNN is used in this study as an alternative training method to minimise the complexity of the extraction of Arabic word root problem.

The chapter has discussed the neural network methodology used in this study. It gave a detailed description of the system prototype, architecture and implementation. Furthermore, it presented the experiments that were used in order to optimise the neural network parameters. The corpus and its preparation also was presented in this chapter.

A number of experiments (**1,600**) were conducted in this chapter to evaluate the suitability of BPNNs for the task of extracting the roots of Arabic words. These experiments were used in order to detect the optimal number of the hidden neurons in the hidden layer, furthermore they are used to evaluate the effect of increasing the number of the hidden neurons on the performance of the BPNNs.

The next chapter deals with the testing and the evaluation of the developed MUAIDI-STEMMER .

Chapter 7

TESTING AND EVALUATION

7.1 Introduction

In the previous two chapters the full functionality of the MUAIDI-STEMMER technique was presented. Chapter 5 covered the **Phase I** of this technique, while Chapter 6 covered the **Phase II**, based on the backpropagation artificial neural network.

This chapter deals with the evaluation of the developed stemmer. Evaluation is an important issue in the development of language technology. In the stemming process, evaluation reflects the stemmer quality and unfolds its effectiveness. The chapter starts by presenting the different evaluation methodologies that are used to evaluate the stemmer algorithms. Then, it presents the experiments and the results results obtained by applying **Phase I** and **Phase II** to the set of testing data. Also, the chapter contains a comparison between the developed stemmer and another stemmer. Finally, it discuses the error types which was unfolded during the evaluation.

7.2 Stemmer Quality Evaluation Methodologies

In literature, there are different measurement methodologies to measure the quality of a stemmer and to evaluate its effectiveness. We can distinguish three kinds of evaluation measurements:

- Recall and Precision Evaluation Method [44].
- Error Rate Evaluation Method.
- Paice's Evaluation Method [113][115][114].

Recall and precision evaluation method is commonly used in information retrieval systems [44]. The main goal of an information retrieval system is to retrieve all the relevant documents. As mentioned in previous chapters, stemmers have a strong impact on the effectiveness of retrieving these relevant documents. Retrieval effectiveness and performance is usually measured by computing the recall and precision. **Recall** is defined to be the ratio of the number of relevant documents retrieved to the total number of relevant documents. **Precision** is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved.

In order to compute the **recall and precision** measurements, the information retrieval system should be built first and this is beyond the scope of this study. Furthermore, this evaluation method does not show the specific type of errors resulting from the stemmer. For all the previous reasons, **recall and precision** evaluation method will not be used to evaluate our developed stemmer.

Error rate evaluation method employs the idea of manually preparing a dataset that contains pairs of words and their correct roots. Then the evaluated stemmer is tested using this dataset. Each extracted root is compared against the provided one. The **error rate** of a stemmer is measured by counting the number of incorrectly stemmed words divided by the total number of

tested words. The mathematical calculation of this evaluation is shown in Equation 7.1.

$$\textbf{Error Rate} = \frac{\textbf{Incorrectly Stemmed Words}}{\textbf{Total Tested Words}} * 100\% \tag{7.1}$$

There are two advantages for using the **error rate** evaluation method. **First,** it permits quantifying the overall **accuracy rate** of a stemmer. The stemmer **accuracy rate** (hereafter refers to as S_{acc}) gives the percentage of correctly stemmed words in the tested dataset. The mathematical calculation of S_{acc} is shown in Equation 7.2.

$$S_{acc} = 100\% - ErrorRate \tag{7.2}$$

Secondly, it allows to discover the type of errors that result from the stemmer. Experiments 1 and 2, which are presented in Subsections 7.3.1 and 7.3.2 consequently, deal with the **error rate** evaluation method.

Paice [113][115][114] presented in 1994 an evaluation method to check the quality of a stemmer outside the information retrieval system. This type of evaluation requires a sample of tested words partitioned into concept groups. This means that the words in each group are related and they share the same stem. The good quality stemmer should conflate all words in a group to the same sharing stem; furthermore this sharing stem should not occur in any other group.

Paice introduced two error types which may occur during the stemming process. These errors are **overstemming** and **understemming**. The definitions of these errors are below. A good quality stemmer should minimise the **overstemming** and **understemming** errors as mush as possible.

UNDERSTEMMING This occurs when two separately derived words belonging to the same conceptual group are stemmed to different roots where they in fact should be. For example, if the word **running** is stemmed to **run**, while the word **ran** is stemmed to **ran** instead of **run**.

OVERSTEMMING This occurs when two separately derived words belonging to different conceptual groups are stemmed to the same root where they in fact should not be. For example, the word **experiment** as well as the word **experience** are stemmed to the stem **experi**.

A summary of the **Paice's evaluation method** is presented here. Many of the equations are adopted directly from [114] and [115].

1- COMPUTING UNDERSTEMMING INDEX (UI)

First, compute the Desired Merge Total **(DMT)**, which is the number of pairs of different possible words in the particular group, and is given by Equation 7.3:

$$DMT_g = 0.5\, n_g\, (n_g - 1) \tag{7.3}$$

where g is an index for the concept group, and n_g is the number of words in the group g.

Second, compute the Global Desired Merge Total **(GDMT)** by summing the Desired Merge Total **(DMT)** values for all groups in the test dataset. This is given by Equation 7.4:

$$GDMT = \sum_{i=1}^{n_g} DMT_i \tag{7.4}$$

After applying the stemmer to the test dataset, some groups still contain two or more distinct stems, incurring **understemming** errors. So, the Unachieved Merge Total **(UMT)** counts the number of *understemming* errors for each group and this is given by Equation 7.5:

$$UMT = 0.5 \sum_{i=1}^{f_g} u_{gi}\, (n_i - u_{gi}) \tag{7.5}$$

where f_g is the number of distinct stems in group g and u_{gi} is the number of instances in group g which are reduced to the stem i.

Now, compute the Global Unachieved Merge Total **(GUMT)** by summing the

Unachieved Merge Total **(UMT)** values for all groups in the test dataset. This is given by Equation 7.6:

$$GUMT = \sum_{i=1}^{n_g} UMT_{gi} \tag{7.6}$$

Finally, compute the **Understemming Index (UI)** by dividing Equation 7.6 by Equation 7.4 as shown in Equation 7.7:

$$UI = \frac{GUMT}{GDMT} \tag{7.7}$$

2- COMPUTING OVERERSTEMMING INDEX (OI)

First, compute the Desired Non-merge Total **(DNT)**, which counts the possible word pairs formed by a member and a non-member word and is given by Equation 7.8:

$$DNT_g = 0.5\, n_g\, (W - n_g) \tag{7.8}$$

where W is the total number of word types in the test dataset.

Second, compute the Global Desired Non-merge Total **(GDNT)** by summing the Desired Non-merge Total **(DNT)** values for all groups in the test dataset. This is given by Equation 7.9:

$$GDNT = \sum_{i=1}^{n_g} DNT_i \tag{7.9}$$

After applying the stemmer to the test dataset, some groups still contain two or more distinct stems, incurring **overstemming** errors. The Unachieved Merge Total **(WMT)** counts the number of **overstemming** errors for each group and is given by Equation 7.10:

$$WMT_s = 0.5 \sum_{i=1}^{f_s} u_{si}\, (n_i - u_{si}) \tag{7.10}$$

where f_s is the number of distinct stems in group g and u_{si} is the number of instances in group g which are reduced to stem i.

Now, compute the Global Unachieved Non-merge Total (**GWMT**) by summing the Unachieved Non-merge Total (**WMT**) values for all groups in the test dataset. This is given by Equation 7.11:

$$GWMT = \sum_{i=1}^{s} WMT_i \tag{7.11}$$

Finally, compute the **Overstemming Index (OI)** by dividing Equation 7.11 by Equation 7.9 as shown in Equation 7.12:

$$OI = \frac{GWMT}{GDNT} \tag{7.12}$$

7.3 Experiments Design

In order to evaluate the effectiveness of the MUAIDI-STEMMER presented in this thesis, a sequence of tests is carried out. The tests are run on two different datasets using two experiments. The first experiment is to evaluate the **Phase I**, while the second experiment is to evaluate the **Phase II**. The evaluation used here is based on **error rate evaluation method**. The two experiments are described in the following subsections.

7.3.1 Experiment 1

The purpose of this experiment is to test the performance of **Phase I** illustrated in Chapter 5. The evaluation method used here is based on **Error rate** evaluation method. In this experiment, the test datasets are extracted from the annotated corpus which was presented in Section 6.3 of Chapter 6.

The size of these test datasets are **214,407** words which have **42,714** word types. The length of these words is ranging between **4** and **12** and all of them have satisfied the pre-conditions for **Phase I** (see Section 5.3.2 of Chapter 5),

which are:

- The number of zeros are three; or
- The number of zeros are two and the root-distance is greater than or equal two.

Table 7.1 shows the distributions of the test datasets used in testing **Phase I.**

Word Length	All Words	Word Types
4	49,376	5,874
5	54,845	9,232
6	43,921	9,745
7	35,392	8,789
8	20,768	5,972
9	7,212	2,233
10	1,985	556
11	774	263
12	134	50
Total	**214,407**	**42,714**

Table 7.1: The Testing Datasets for **Phase I**

Every word length group is tested independently. The accuracy S_{acc} for each group is calculated. This is done by matching the resultant root against the roots in the **Common Dictionary** (see Section 4.2.1 of Chapter 4).

If the extracted root is found in the **Common Dictionary**, then the combination of the resultant prefix and the suffix is checked against the **prefix-suffix** list (see Section B.2 of Appendix B) that corresponding to the length of the word. If this combination is founded, the the resultant root is considered as a correct root. Otherwise, it is counted as an error. This matching procedure seems to give the correct root most of the time. More formally, given an Arabic word W, and the segmentation of W is $W = P + R + S$, then if P is a valid prefix, R a valid root and S a valid suffix, and if the combination of R and S is

found in the **prefix-suffix** list, then R is considered as a correct root.

The reason behind the two checks above, is that the checking against the roots dictionary alone is not always represent the accurate measurement, especially for the words that may return more than one valid root. For example, suppose that the stemmer stemmed the Arabic word [مطرقة] (mTrqt,"hammer") as [مطر] (mTr,"rain"). The resultant root [مطر](mTr) is a valid root and it is in the **Common Dictionary**. But, unfortunately this is not the correct root for the tested word. The correct one is [طرق] (Trq,"to hammer"). Table 7.2 shows the resultant two roots and their corresponding prefixes and suffixes.

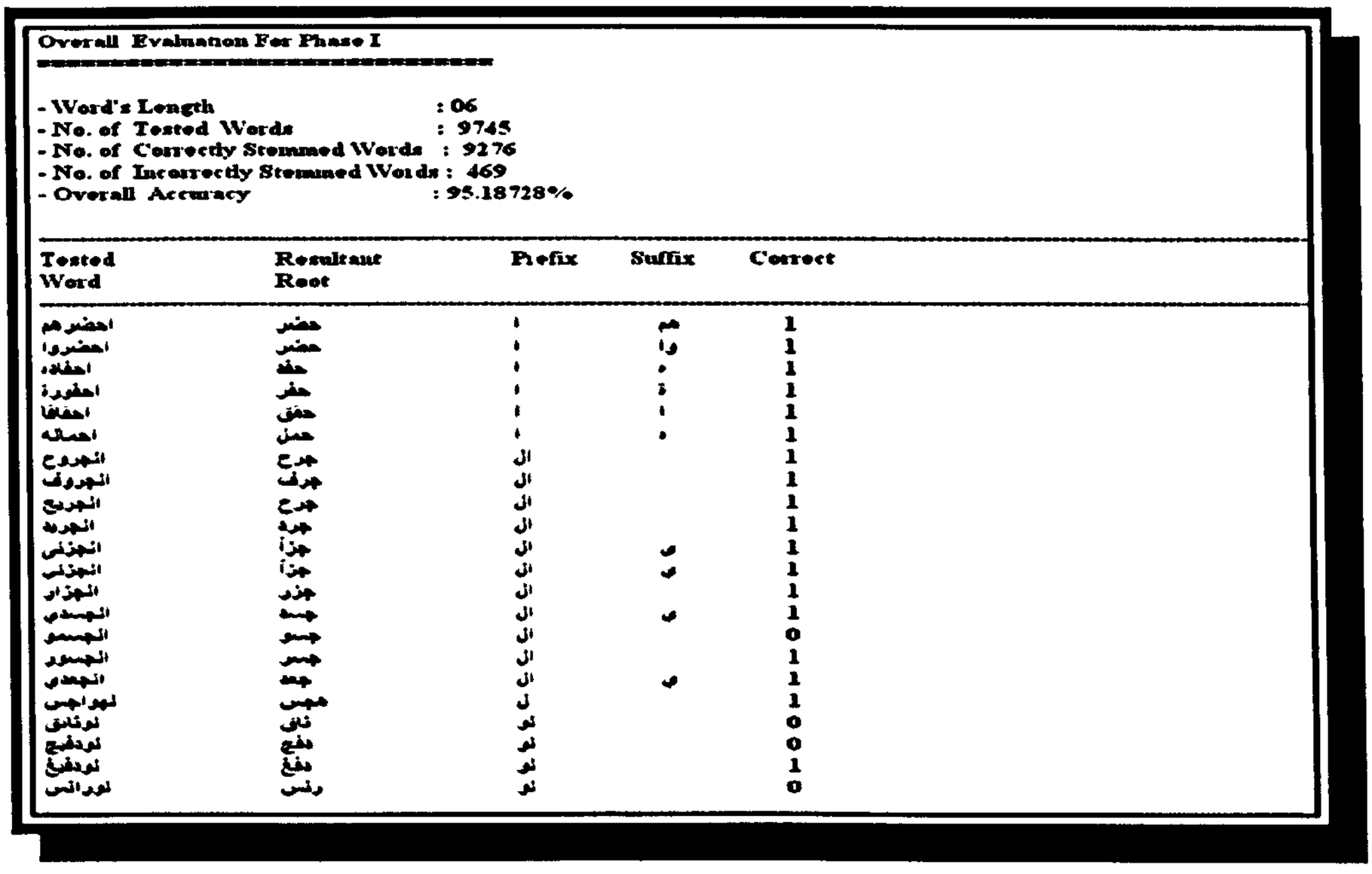
Word	Root	Prefix	suffix
مطرقة	مطر	NULL	قة
مطرقة	طرق	م	ة

Table 7.2: An Example of Two Roots for the Same Word

Now, if the combination of the prefix and the suffix (NULL+[قة]) of the first root is checked against the **prefix-suffix** list, then this combination is not found. In consequence, the root [مطر] is considered as an incorrect root. On the other hand, the root [طرق] is considered as a correct root for the tested word, since, its prefix and suffix combination ([م]+[ة]) is found in the **prefix-suffix** list.

Figure 7.1 shows an example of an output file has some examples of tested words belong to length category **6**. The output file is formatted in four columns. The left-most column lists all the tested words. The second column lists the resultant roots from the system. The third and fourth columns list the extracted prefixes and suffixes, respectively, while, the right-most column, which is the important one in this evaluation, lists if the resultant root is the correct root (1) or not (0).

The top lines of the output file in Figure 7.1 view some statistics in behave of the tested data. This includes the category number, total number of tested



Overall Evaluation For Phase I

- Word's Length : 06
- No. of Tested Words : 9745
- No. of Correctly Stemmed Words : 9276
- No. of Incorrectly Stemmed Words : 469
- Overall Accuracy : 95.18728%

Tested Word	Resultant Root	Prefix	Suffix	Correct
احضرهم	حضر	ا	هم	1
احضروا	حضر	ا	وا	1
احفاه	حفه	ا	ه	1
احفورة	حفر	ا	ة	1
احفافا	حفف	ا	ا	1
احماله	حمل	ا	ه	1
الجروح	جرح	ال		1
الجروف	جرف	ال		1
الجريح	جرح	ال		1
الجرة	جره	ال		1
الجزئى	جزأ	ال	ى	1
الجزئى	جزأ	ال	ى	1
الجزار	جزر	ال		1
الجسدى	جسد	ال	ى	1
الجسمو	جسو	ال		0
الجسور	جسر	ال		1
الجهدى	جهد	ال	ى	1
لهواجس	هجس	ل		1
لوثاق	ثاق	لو		0
لودفيج	دفج	لو		0
لودفيغ	دفغ	لو		1
لورانس	رنس	لو		0

Figure 7.1: Some Examples of Tested Words Belong to Length Category 6

words, total number of correctly stemmed words, total number of incorrectly stemmed words, and finally, the overall accuracy.

Every tested category has its own output file. The accuracies in these files are amalgamated in Table 7.3.

Word's Length	Word Types	Correctly Stemmed	Performance S_{acc}
4	5,874	5,306	90.33%
5	9,232	8,676	93.98%
6	9,745	8,940	91.74%
7	8,789	8,102	92.18%
8	5,972	5,612	93.97%
9	2,233	2,118	94.85%
10	556	545	98.02%
11	263	263	100.00%
12	50	50	100.00%
Total	**42,714**	**39,612**	**92.74%**

Table 7.3: Stemming Accuracy for all Category Length in **Phase I**

The obtained experimental results in Table 7.3 show that MUAIDI-STEMMER in its **Phase I** produced correctly **39,612** roots from **42,714** in the given test datasets with an accuracy rate **92.74%**.

The error analysis of **Phase I** unfolds that many errors happened when the root is a **hamzated** root. A **hamzated** root has a **hamza** [ء] in one of its consonants. Examples of these errors are shown in Table 7.4.

Tested Word	Transliteration	Produced Root	Correct Root
مؤرخ	mwrĤ	ؤرخ	أرخ
أخطئ	ĀĤTÃ	خطئ	خطأ
صفاء	SfA’	صفء	صفا

Table 7.4: Errors Come Due **Hamzated**

The **hamzated** root errors presented above establish that improvement of the developed stemmer is possible. One possibility of improvement is by adding a normalisation rule for the **hamzated** resultant root. This rule just changes the shape of **hamza** to follow the correct writing grammar in Arabic. Figure 7.2 illustrates the normalisation rule.

```
if (a root has [ؤ] or [ئ] letter) then
    replace this letter by [إ].
if (a root has [ء] letter) then
    replace this letter by [ا].
```

Figure 7.2: An Improvement Step

The first rule in Figure 7.2 is to replace the letters [ؤ](w̄) and [ئ](Ã) to the letter [إ](Ā), while the second one is to replace the letter [ء](’) to the letter [ا](A).

In order to check the effect of this normalisation rule on the performance of **Phase I**, the developed stemmer was run again on the same test datasets that were used at the beginning of this experiment. **Phase I** produced **40,638** correct roots from **42,714** in the test datasets. The total errors were reduced

to 2,076 and the accuracy performance of the stemmer is improved to 95.14%. The new accuracies after applying the normalisation rule are shown in Table 7.5.

Word's Length	Word Types	Correctly Stemmed	Performance S_{acc}
4	5,874	5,437	92.56%
5	9,232	8,886	96.25%
6	9,745	9,276	95.19%
7	8,789	8,216	93.48%
8	5,972	5,765	96.53%
9	2,233	2,189	98.03%
10	556	556	100.00%
11	263	263	100.00%
12	50	50	100.00%
Total	**42,714**	**40,638**	**95.14%**

Table 7.5: Stemming Accuracy for all Category Length in **Phase I** After Applying the Normalisation Rule

7.3.2 Experiment 2

The aim of this experiment is to investigate the viability of applying ANNs approach as a tool for extracting Arabic word roots. This is done by applying the test datasets for the trained BPNNs illustrated in the previous chapter. The test datasets used here are different from those that were used in the training processes, in order to evaluate the performance of BPNNs on unseen data. The ANNs used in this experiment represent the best BPNNs developed thus far in terms of minimising the mean squared error (MSE) over the datasets used in the training processes.

The test datasets consist of **25%** of the total amount of data from which **75%** was used for training the ANNs (see Section 6.3). The total size of the testing datasets are **5,906** word types with lengths ranging between **4** and **11** letters. Table 7.6 shows the distributions of these datasets.

BPNN	Training Words	Testing Words
$BPNN_4$	3,248	1,083
$BPNN_5$	5,789	1,930
$BPNN_6$	3,922	1,307
$BPNN_7$	2,177	725
$BPNN_8$	1,076	358
$BPNN_9$	1,184	395
$BPNN_{10}$	284	94
$BPNN_{11}$	44	14
Total	**17,724**	**5,906**

Table 7.6: The Training and Testing Datasets Used in **Phase II**

Every developed BPNN was tested independently. The obtained experimental and testing results are analysed with respect to two elements: ANN architectures, and the ANN performances based on their generalisation ability to identify the correct root class for the unseen Arabic word.

1- The ANN Architectures

As presented in Section 6.5.3.1 of Chapter 6, different ANNs architectures have been attempted in order to find the optimal architectures for all the developed BPNNs. This was done by varying the number of hidden neurons in the hidden layer. We ran **1,600** experiments to achieve this goal. The summarisations of these experiments are depicted in Figure 7.3. Figure 7.3 has eight sub-figures, every figure represented one of the developed BPNNs. The x-axis represents the number of hidden neurons, while the y-axis represents the average of errors obtained during the training of BPNN.

By comparing the results of the different architectures across each of the line graphs shown in Figures 7.3, it was found that increasing the number of hidden neurons is not always improve the performance of the network. The full details of the experiment results are tabulated in the Appendix F.

The BPNNs showed acceptable number of hidden neurons except for one case in $BPNN_4$. The $BPNN_4$ has **40** neurons as the best number of neurons in the hidden layer. This number seems too high with respect to the number of the

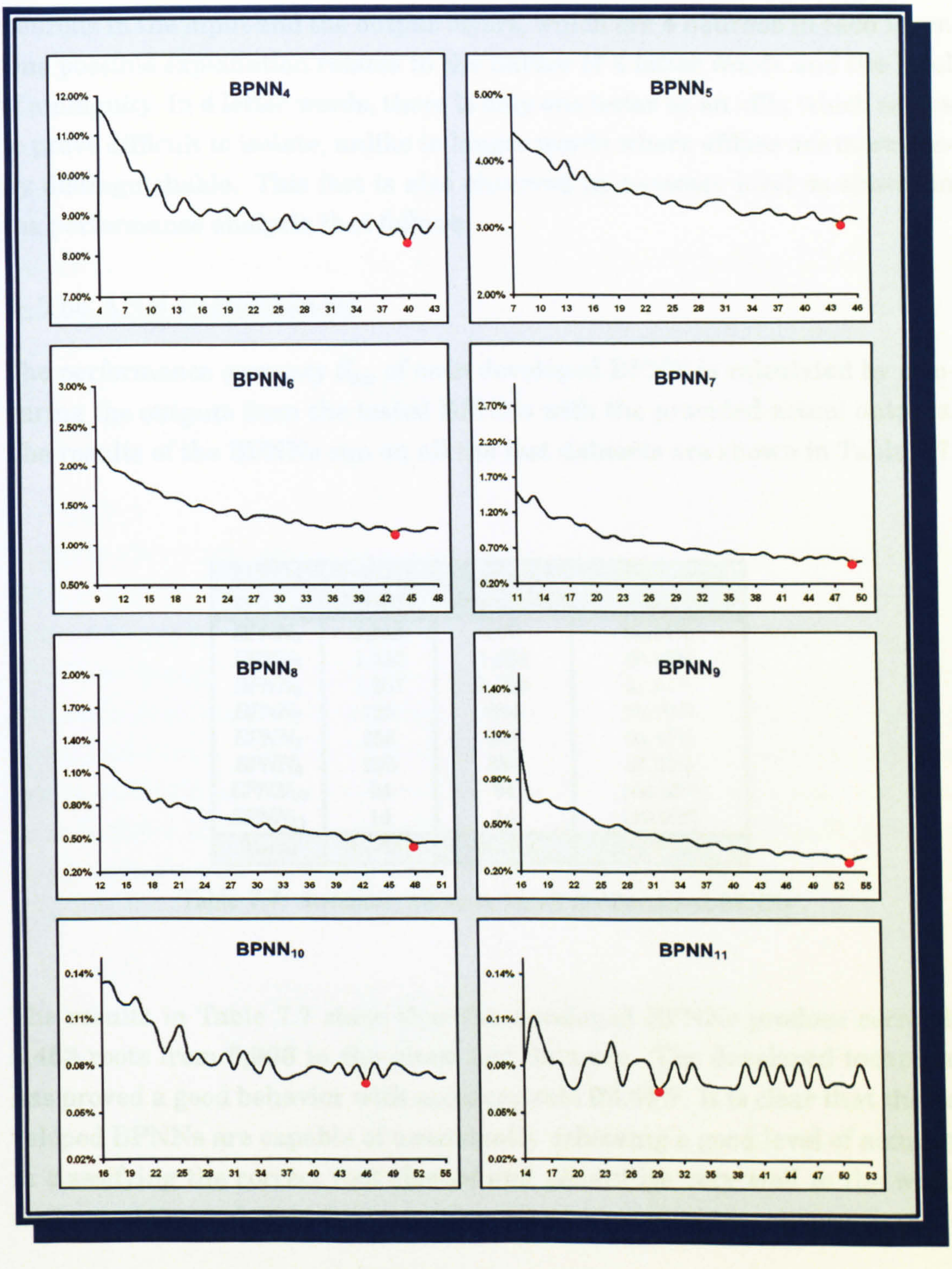


Figure 7.3: Stemming Accuracy for all Category Length in **Phase I**

neurons in the input and the output layers, which are **4** neurons in each layer. One possible explanation relates to the nature of **4** letter words and the level of ambiguity. In **4** letter words, there is only one letter as an affix which seems to prove difficult to isolate, unlike in longer words where affixes are more easily distinguishable. This fact is also observed in accuracy level as shown in the performance analysis that follows.

2- The ANN Performances

The performance accuracy S_{acc} of each developed BPNN is calculated by comparing the outputs from the tested BPNNs with the provided actual outputs. The results of the BPNNs run on all the test datasets are shown in Table 7.7.

BPNN	Testing Words	Correctly Stemmed	Performance S_{acc}
$BPNN_4$	1,083	977	90.21%
$BPNN_5$	1,930	1,754	90.88%
$BPNN_6$	1,307	1,200	91.81%
$BPNN_7$	725	694	95.72%
$BPNN_8$	358	343	95.81%
$BPNN_9$	395	384	97.22%
$BPNN_{10}$	94	94	100.00%
$BPNN_{11}$	14	14	100.00%
Total	**5,906**	**5,460**	**92.44%**

Table 7.7: Stemming Accuracy for all the Developed BPNNs

The results in Table 7.7 show that the developed BPNNs produce correctly **5,458** roots from **5,906** in the given test datasets. The developed technique has proved a good behavior with accuracy rate **92.44%**. It is clear that the developed BPNNs are capable of consistently achieving a good level of accuracy in classifying the correct root classes and generalise very well to the words that are not seen during the training process.

We observe also that there is a strong relationship between the accuracy and the length of words in each test set. This can be observed clearly from Ta-

ble 7.7. The accuracy increases significantly as the length of words increases. This is due to the nature of the neural networks, when there are sufficient inputs and features, the efficiency of the neural networks is increased.

7.3.3 Experiment 3

Before evaluating the MUAIDI-STEMMER against the **Paice evaluation method** which was presented in Section 7.2 of this chapter, it is worth understanding this method more clearly. Also, it is worth checking the viability of this evaluation method for Arabic language. In order to achieve this, two examples are discussed below.

Example 1

Suppose there are seven Arabic words shown in the second column of Table 7.8 falling into two distinct conceptual groups. All the words with the same root are joined in one group. The first five words share the root [سعد] (s'd,"to be happy"), while the last two words share the root [عود]('wd," to accustom"). Also, suppose these words are tested against two virtual stemmers $Stem_1$ and $Stem_2$. The resultant roots from these stemmers are shown in the fourth and fifth columns of Table 7.8, respectively. The incorrect resultant roots in these columns are underlined and coloured by red. The gold standard (truth set) for this experiment is the combination of the second and third columns, respectively.

Group	Tested Word	Correct Root	$Stem_1$	$Stem_2$
1	السعودية	سعد	عود	عود
1	سعادات	سعد	سعد	سعد
1	سعيدة	سعد	سعد	سعد
1	مسعود	سعد	سعد	عود
1	والسعادة	سعد	سعد	سعد
2	العودات	عود	عود	عود
2	سأعوده	عود	عود	عود

Table 7.8: A Data Example for Paice Evaluation (1)

By comparing the resultant roots from the two stemmers by the gold stan-

dard, it is obvious that $Stem_1$ produces more correct roots than $Stem_2$. In consequence, $Stem_1$ has a good quality than $Stem_2$. The accuracy of $Stem_1$ is $S_{acc_1} = 85.7\%$ while the accuracy of $Stem_2$ is $S_{acc_2} = 71.4\%$. Now, we will follow the **Paice evaluation method** steps to comapre the two stemmers.

1. Use Equation 7.4 and Equation 7.6 to compute the **GDMT** and **GUMT** for the two stemmers. The resultant computations are shown in Tables 7.9 and 7.10 respectively.

Group	$Stem_1$ DMT_1	$Stem_2$ DMT_2
1	10	10
2	1	1
GDMT	**11**	**11**

Table 7.9: Calculating GDMT for $Stem_1$ & $Stem_2$

Group	$Stem_1$ UMT_1	$Stem_2$ UMT_2
1	4	6
2	0	0
GUMT	**4**	**6**

Table 7.10: Calculating GUMT for $Stem_1$ & $Stem_2$

2. Use Equation 7.7 to compute the **Understemming Index** for the two stemmers. The resultant values are shown in Equations 7.13 and 7.14 respectively.

$$UI(Stem_1) = \frac{GUMT_1}{GDMT_1} = \frac{4}{11} = 0.36 \tag{7.13}$$

$$UI(Stem_2) = \frac{GUMT_2}{GDMT_2} = \frac{6}{11} = 0.55 \tag{7.14}$$

3. Use Equation 7.9 and Equation 7.11 to compute the **GDNT** and **GWMT** for the two stemmers. The resultant computations are shown in Tables 7.11 and 7.12 respectively.

Group	$Stem_1$ DNT_1	$Stem_2$ DNT_2
1	5	5
2	5	5
GDNT	**10**	**10**

Table 7.11: Calculating GDNT for $Stem_1$ & $Stem_2$

Root	$Stem_1$ WMT_1	$Stem_2$ WMT_2
سعد	0	0
عود	2	4
GWMT	**2**	**4**

Table 7.12: Calculating GWMT for $Stem_1$ & $Stem_2$

4. Use Equation 7.12 to compute the **overstemming index** for the two stemmers. The resultant values are shown in Equations 7.15 and 7.16 respectively.

$$OI(Stem_1) = \frac{GWMT_1}{GDNT_1} = \frac{2}{10} = 0.20 \tag{7.15}$$

$$OI(Stem_2) = \frac{GWMT_2}{GDNT_2} = \frac{4}{10} = 0.40 \tag{7.16}$$

By comparing the two values of **UI** and **OI** for the both stemmers, it is clear that the $Stem_1$ is better than $Stem_2$ in terms of **understemming** and **overstemming**. In consequence, the performance of $Stem_1$ is better than $Stem_2$ and this is reflected in our initial assumption that $Stem_1$ has a good quality than $Stem_2$. Fiqure 7.4 shows a snapshot of all the obtained results.

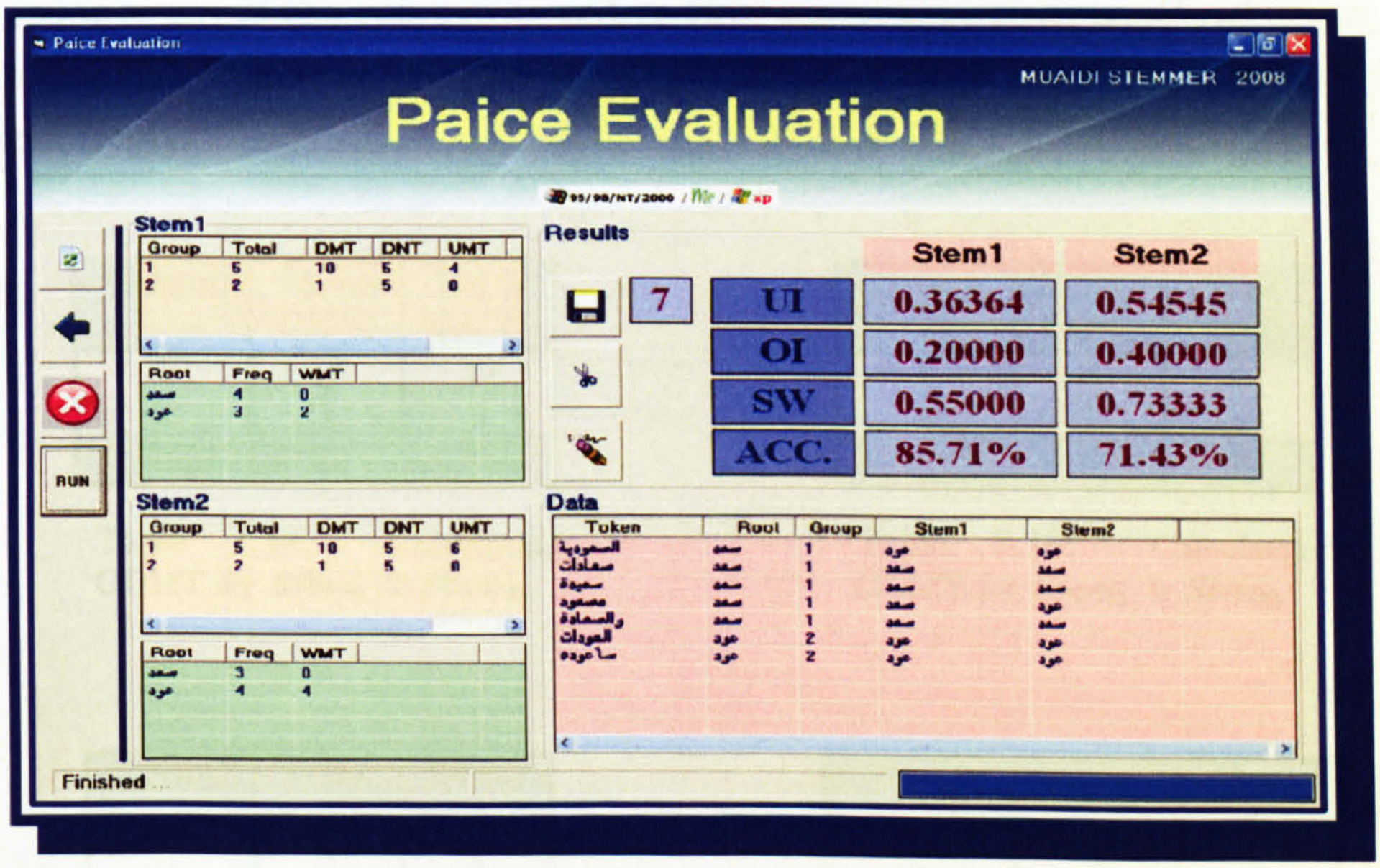


Figure 7.4: Snapshot of Paice Evaluation

Example 2

This example uses the same data that presented in Example 1 above except in this example it is assumed that the $Stem_1$ is produced two incorrect roots as what $Stem_2$ was did. In this situation the performances of $Stem_1$ and $Stem_2$

are equivalent. The data of this experiment is shown in Table 7.13. Note that some roots in the columns four and five are underlined and coloured by red to indicate the incorrect roots.

Group	Tested Word	Correct Root	$Stem_1$	$Stem_2$
1	السعودية	سعد	سعد	عود
1	سعادات	سعد	سعد	سعد
1	سعيدة	سعد	سعد	سعد
1	مسعود	سعد	سعد	عود
1	والسعادة	سعد	عود	سعد
2	العودات	عود	عود	عود
2	سأعوده	عود	سعد	عود

Table 7.13: A Data Example for Paice Evaluation (2)

In order to avoid the retraction of the **Paice's evaluation method** steps, the final results obtained are presented in Tables 7.14, 7.15, 7.16, 7.17 and 7.18 respectively.

Group	$Stem_1$ DMT_1	$Stem_2$ DMT_2
1	10	10
2	1	1
GDMT	**11**	**11**

Table 7.14: Calculating GDMT for $Stem_1$ & $Stem_2$

Group	$Stem_1$ UMT_1	$Stem_2$ UMT_2
1	4	6
2	1	0
GUMT	**5**	**6**

Table 7.15: Calculating GUMT for $Stem_1$ & $Stem_2$

Group	$Stem_1$ DNT_1	$Stem_2$ DNT_2
1	5	5
2	5	5
GDNT	**10**	**10**

Table 7.16: Calculating GDNT for $Stem_1$ & $Stem_2$

Root	$Stem_1$ WMT_1	$Stem_2$ WMT_2
سعد	4	0
عود	1	4
GWMT	**5**	**4**

Table 7.17: Calculating GWMT for $Stem_1$ & $Stem_2$

Stemmer	*OI*	*UI*
$Stem_1$	0.50	0.45
$Stem_2$	0.40	0.55

Table 7.18: The Summary of **OI** and **UI** for the Two Stemmers

Table 7.18 shows clearly that the $Stem_1$ is better in terms of **understemming** but it is worse in terms of **overstemming**. Now, there are two different measures quantifying the performance of each suggested stemmer. So, which measure is more important than the other? **Paice** does not offer an answer for this question. But, he suggests a way to combine the two measures. This is done by the so-called **stemming weight (SW)**. The **SW** is calculated by dividing the **overstemming index (OI)** by the **understemming index (UI)**, as shown in Equation 7.17. The higher **SW** value indicates a stronger stemmer.

$$SW = \frac{OI}{UI} \tag{7.17}$$

Equation 7.18 shows the **stemming weight** value for $Stem_1$, while 7.19 shows the **stemming weight** value for $Stem_2$. By comparing these two values, it is clear that $Stem_1$ has the higher **SW** value, which indicates that the performance of $Stem_1$ is better than $Stem_2$. This final result is deny our initial assumption fact which that $Stem_1$ has the same quality as $Stem_2$.

$$SW(Stem_1) = \frac{OI(Stem_1)}{UI(Stem_1)} = \frac{0.50}{0.45} = 1.1 \tag{7.18}$$

$$SW(Stem_2) = \frac{OI(Stem_2)}{UI(Stem_2)} = \frac{0.55}{0.40} = 0.73 \tag{7.19}$$

To be more accurate we run another seven experiments using the same seven words and the gold standard which were presented in Example 1 above. The number of correct roots for the both stemmers is vary from experiment to experiment. Table 7.19 summerizes the number of correct roots, the accuracy value, OI, UI, and SW of each experiment.

Exp.	Stemmer	Correct Roots	Accuracy	OI	UI	SW
3	$Stem_1$	7	100%	0	0	∞
	$Stem_2$	7	100%	0	0	∞
4	$Stem_1$	2	28.57	1	0	∞
	$Stem_2$	7	100%	0	0	∞
5	$Stem_1$	0	0.0%	0	0	∞
	$Stem_2$	7	100%	0	0	∞
6	$Stem_1$	0	0.0%	0	0	∞
	$Stem_2$	0	0.0%	0	0	∞
7	$Stem_1$	4	57.14%	0.6	0.55	1.1
	$Stem_2$	3	42.86%	0.8	0.36	2.2
8	$Stem_1$	5	71.43%	0.4	0.55	0.73
	$Stem_2$	2	28.57%	0.5	0.45	1.1
9	$Stem_1$	2	28.57%	1	0	∞
	$Stem_2$	6	85.71%	0.2	0.36	0.56

Table 7.19: Different Experiments to Check the Viability of Paice Evaluation Method for Arabic Language

Table 7.19 clearly shows that there is a conflict between the accuracy values and the Paice measurement parameters. In Experiment 3 the accuracy of the two stemmers is the same, and the Paice measurement parameters are equal, which yields to accept this result. The contrast is in Experiment 6, according to Paice, a perfect stemmer would have a UI and OI value of zero, in this experiment despite the two stemmers have a UI and OI value of zero, they have zero accuracies. In Experiments 4, 5 and 9 the accuracy of $Stem_2$ is betters than $Stem_1$, but unfortunately, the Paice measurement parameters deny these results. Finally, in Experiments 7 and 8 the accuracy of $Stem_1$ is betters than $Stem_1$, but also, the Paice measurement parameters deny these results.

In conclusion, the Paice method is considered with grouping the stems from the perspective of information retrieval system rather than the correctness of the stem itself. For this reason and for the results obtained in Table 7.19 above, we can claim that Paice evaluation method is not viable for Arabic language in order to compare the accuracy performance of two different stemming algorithms.

7.3.4 Experiment 4

The aim of this experiment is to compare the performance of MUAIDI-STEMMER against KHOJA-STEMMER. KHOJA-STEMMER was discussed in Section 3.5 of Chapter 3, the reason behind this choosing, is that KHOJA-STEMMER is the most common cited **root-stemming** algorithm and it is showed superiority over many stemmers in the field. Furthermore, the Java version of this algorithm is already published in Khoja homepage[1].

To compare the performance of the both stemmers, a gold standard contains a sample of **1000** Arabic words is produced for this purpose. We evaluated the two stemmers against the gold standard, and computed the accuracy values.

To be fair, the gold standard was created manually by an Arabic language expert[2]. Most of these words are extracted from the Quran, the holy book of Islam. For each word, the corresponding root is illustrated. Figure 7.5 shows a sample of the gold standard which is grouped according to the word roots.

Group	Root	Words			
1	صدق	صداقه مصدقين فلصدقتهم	صادقات مصدقات صدقناهم	صادقين يصدقون الصديق	صديقات فيصدقه متصدقات
2	خرج	نستخرجه مستخرج	وسيخرجوهن اخرجتك	وسيخرجاهما نستخرجهن	استخرجوهم واخرجوكم

Figure 7.5: Some Words and their Roots in the Gold Standard

The results of the comparison of MUAIDI-STEMMER and KHOJA-STEMMER are shown in Figure 7.6.

The statistics in Figure 7.6 show that the total number of words processes

[1] http://zeus.cs.pacificu.edu/shereen/research.htm#stemming

[2] The author gratefully acknowledges the assistance of Fahed Ashour; email: fahed.ashour@gsaa.uni-halle.de. Martin Luther University, Halle, Germany.

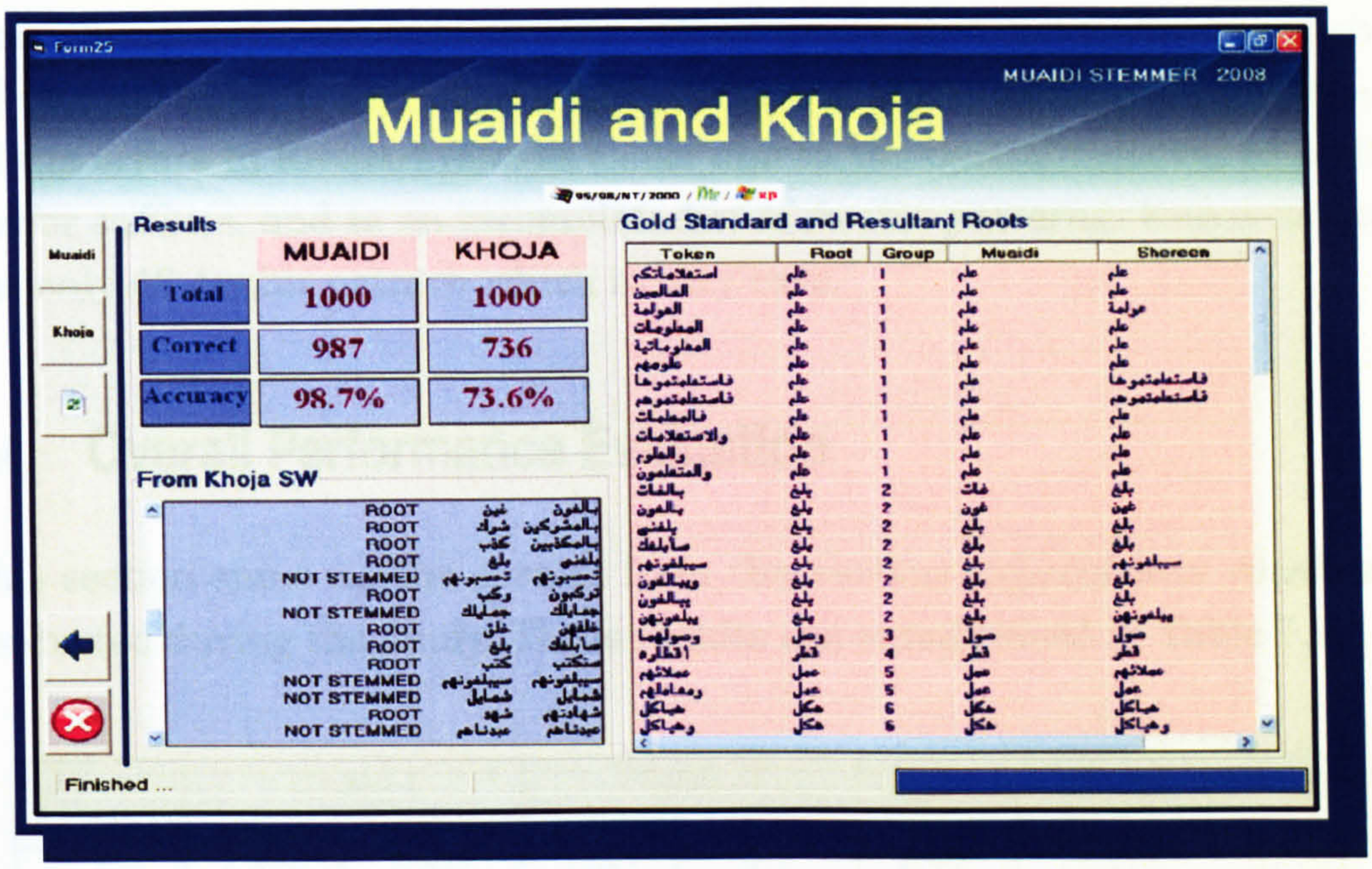


Figure 7.6: Snapshot of A Comparison of MUAIDI-STEMMER and KHOJA-STEMMER

is **1,000** word types. MUAIDI-STEMMER successfully stemmed **987** with accuracy **98.7%**, while KHOJA-STEMMER successfully stemmed **736** with accuracy **73.6%**. Looking at the unstemmed words produced by KHOJA-STEMMER , it is apparent that this stemmer is not able to remove the prefixes and/or the suffixes of some words and this leads to it generate the incorrect roots. For example, the word [عملائهم] ('mlAÃhm,"their customers") should be stemmed to give the root [عمل] ('ml). Unfortunately, KHOJA-STEMMER fails in stemming this word, it starts the stemming process by removing the suffix [هم] (hm,"their") from the word, producing the stem [عملائ] ('mlAÃ), which has a [ئ](Ã,"Hamza on Alef Maqsura") as a last letter. When KHOJA-STEMMER tries to match the resultant stem with the triliteral Patterns list, it fails since there is no pattern has the paradigm [فعلائ] (f'lA'). The correct pattern is [فعلاء] (f'lA') which is found in the triliteral Patterns list.

KHOJA-STEMMER fails to stem any word has the pattern [فياعل] (fyA'l) since this pattern is not found in the pattern list. For example, all the words [شمايل](ŝmAyl), [هياكل](hyAkl), and [جمايلك] (jmAylk) are stemmed incorrectly.

Also, KHOJA-STEMMER fails to stem some words that have the conjunctions [و] (w,"and"), [ف](f,"therefore"), [ل](l,"to"), and [ك](k,"as/like"). In summary, all the errors in KHOJA-STEMMER cause due to the partial removal of prefixes and/or suffixes, and to an incomplete list of Arabic patterns. KHOJA-STEMMER has only **46** Arabic pattern which is very rare.

7.4 Overall Performance Evaluation

This section sums up the results from the various experiments which were conducted during the study. These results are summarised in Table 7.20.

Word Length	Phase I		Phase II		Total		
	Types	Correct	Types	Correct	Types	Corrects	%
4	5,874	5,437	1,083	977	6,957	6,414	92.19%
5	9,232	8,886	1,930	1,754	11,162	10,640	95.32%
6	9,745	9,276	1,307	1,200	11,052	10,476	94.78%
7	8,789	8,216	725	694	9,514	8,910	93.65%
8	5,972	5,765	358	343	6,330	6,108	96.49%
9	2,233	2,189	395	384	2,628	2,573	97.91%
10	556	556	94	94	650	650	99.85%
11	263	263	14	14	277	277	100.00%
12	50	50	0	0	50	50	100.00%
Total	**42,714**	**40,638**	**5,906**	**5,460**	**48,620**	**46,096**	**94.81%**

Table 7.20: The Overall Results

The number of correct roots is **46,096** from **48,620** with accuracy **94.81%**. The overall results obtained in all of the experiments are very encouraging as that they show the possibility of using the computational approach for extracting the roots of Arabic words.

7.5 Error Analysis

The error analysis presented in this section is conducted by analysing the results of both phases for the developed MUAIDI-STEMMER . The errors obtained are classified into the following classes:

there are no standard rules to identify these nouns. Table 7.22 shows some examples for this type of errors.

Length	Word	Transliteration	Translation
4	مريم	mrym	lady name
5	ليبيا	lybyA	country name
7	الخرطوم	AlĤrTwm	city name

Table 7.22: Errors Come From Proper Nouns

- As mentioned in Section 4.5.2 of Chapter 4, the formation of Arabic words is not usually a straightforward process. Some triliteral roots lose one of their three letters during the morphological inflection. When such words are stemmed, the incorrect roots is produced.

- In some cases the root-letters are modified or deleted during the formation process of the Arabic words as mentioned in Section 4.5.2 of Chapter 4. This is done especially in the weak verb roots. For example, the MUAIDI-STEMMER stemmed the word [مقالة] (mqAlt, "an article") as [قال](qAl) while the correct root is [قول](qwl). In this example, the second letter of the root [و](w) is changed to the letter [ا](A).

From the linguistic analysis of the roots, patterns, and affixes that carried out in this study, some rule were identified rules to solve this problem. Figure 7.7 summarises these rules. These rules were not implemented as part of MUAIDI-STEMMER since extensive experiments would be needed to validate them.

Let C_1, C_2, and C_3 are the letters of the extracted root.

1. **If** C_2= [ا](A) and C_1 belongs to the letters of the word [سألتمونيها](*sĀltmwnyhA*), **then** C_2= [و](w) **else** C_2= [ي](y).
2. **If** C_3= [و](w) **then** C_3= [ي](y).
3. **If** C_3= [ي](y) **then** C_3= [و](w).

Figure 7.7: Some Rules to Correct the Extracted Root

- Some of the Arabic words have not a triliteral roots. The Arabic word [زلزالا] has a quadrilateral root [زلزل]. These types of roots are out of the focus of this research.

- The final class of errors comes from the multiword expressions [التعابير المركبة], in this type two words appear as one single term, in which our stemmer failed to stem such terms. For example, the term [عبدالرحمن] ('bdAlrHmn), it consists of two different words [عبد] ('bd) and [الرحمن] (AlrHmn).

7.6 Summary

The main aim of this chapter was to evaluate the performance of the developed MUAIDI-STEMMER . Three different types of evaluation methods were discussed. These are: **Recall and Precision**, **Error Rate**, and **Paice** evaluation methods.

From the experiments illustrated in this chapter, it is clear that the overall accuracy for the MUAIDI-STEMMER is **94.81%**. This accuracy gives satisfactory results and proves to be a promising stemmer. The accuracy for **Phase I** is **95.14%**, while it is **92.44%** for **Phase II**. These acurracies were calculated based on the **error rate evaluation method**.

Also, MUAIDI-STEMMER was compared against KHOJA-STEMMER . The comparison was done using accuracy rate evaluation method. The results show that MUAIDI-STEMMER makes less errors than KHOJA-STEMMER . The accuracy of MUAIDI-STEMMER is **98.7%**, while the accuracy of KHOJA-STEMMER is **73.6%**. The comparison clearly indicates that MUAIDI-STEMMER developed in this work is more effective than KHOJA-STEMMER .

By running many experiment, this chapter proved that the **Paice** evaluation method is not viable for Arabic language in order to compare the accuracy performance of two different stemming algorithms.

In conclusion, the overall results obtained in all the experiments are very encouraging and they point to the possibility of using the computational and neural network approaches for the Arabic word roots extraction problem.

The next chapter which is the final chapter presents the conclusion of this study.

Chapter 7

Chapter 8

CONCLUSION

8.1 Introduction

This chapter brings us to the end of our long journey through developing MUAIDI-STEMMER technique for Arabic language. The purpose of this chapter is to summarise the achievement which has been made by this study. This includes a review of the research design. It is then followed by the main findings. Based on these findings, a number of recommendations are represented in this chapter. Finally, the chapter ends with some future work which could be carried out as a continuation of this study.

8.2 Study Summary

Arabic is a morphologically complex language. The language uses both inflectional and derivational morphologies. As a result of these types of morphology, a single word in the language yields a number of variant forms. These word form variations can have a strong impact on the effectiveness of the information retrieval systems.

Most existing stemming algorithms for Arabic language rely on rule-based systems, and many of them refer to a lookup table of patterns and roots. In

consequence, this requires a large storage space to store these tables, and time to access the demand information. Due to the above, there is no standard root-stemming algorithm for Arabic language.

The limitations of the current Arabic stemming methods have motivated this author to investigate a novel approach to be used in extracting the word roots of Arabic language. This approach is a computational approach, which avoids having a list of the root and pattern of each word in the language. Furthermore, it has no set linguistic rules.

The computational approach based on this thesis tries to exploit the numerical relations between letters that are identified by Arabic linguists to be affixes. The linguistic studies showed that these letters appear as affixes with different frequencies. Extensive root and affix analysis showed further numerical relation between affixes, root positions and the frequency of affix letters. The developed approach exploits these facts by assigning numerical values that identifies the class of each Arabic letter. These numeric values are then used in encoding a given Arabic word to be presented as an input to the developed stemmer.

The developed stemmer was conducted in two phases: Phase I and Phase II. The next two subsections have summarised these two phases.

8.2.1 Phase I

Phase I is based on the analysis of Arabic patterns and the affix letters. In this phase we assigned a weight value to every letter in the tested word then multiply these values by the score values. The weight values are based on the frequency analysis of Arabic affix letters, while the score values are based on the analysis of the Arabic patterns, which show the possibility of the letter to be an affix letter. If there are three zeros in the resultant multiplications then the corresponding letters of these zeros represent the root-letters.

If there are just two zeros in the resultant multiplications, then the root-

distance is computed, which counts the gap between these zeros (number of letters). If there are two letters or more, then one of these letters is a root-letter, which is represent the minimum weight value.

8.2.2 Phase II

Phase II is based on artificial neural networks trained by the backpropagation learning rule. In this developed phase, we formulated the root extraction problem as a classification problem and the neural network as a classifier. We decomposed the Arabic root extraction complex task into a number of simple tasks. We assigned a specific backpropagation neural network for each task. The term task refers to the Arabic word group that has the same letter length. We denoted this approach as **a multi-backpropagation neural network.** This approach is developed as an alternative training method for backpropagation neural network to minimise the neural complexity. Nine backpropagation neural networks were developed and every word group has its own neural network.

8.3 Study Findings

As was pointed in Chapter 1, the main aim of the study is to develop a stemming algorithm to extract the roots of Arabic words based on a computational and artificial neural networks approaches. So, there were two research questions which were asked at the beginning of Section 1.4 of Chapter 1. The research questions were as follows:

1. **Can we identify and exploit numerical relations between letters for Arabic root extraction ?.**

   To answer this question an extensive linguistic analysis of patterns and affixes in Arabic language were carried out. As a result of the analysis, we found some numerical relations that can enable to extract the root of a given word (**Phase I**). The results of the evaluation show that the accuracy of **Phase I** is **95.14%**.

2. **Can we use the artificial neural networks as a tool to extract the roots of Arabic words ?.**

   In answering this question a multi-backpropagation neural networks have been developed. These neural networks were developed based on experimentation and extensively tested (**Phase II**). The evaluation experiments that were carried out to evaluate the performance of the **Phase II** show that the accuracy is **92.44%**.

The overall developed modules (i.e. **Phase I** and **Phase II**) exhibit accuracy **94.81%** over a corpus counts of **48,620** words. The result is very encouraging and provided a good quality and accurate stemming algorithm, as a consequence, they pointed to the possibility of using the computational and neural network approaches for the Arabic word roots extraction problem.

8.4 Study Contributions

The main contribution that can be drawn from this study is the application of a computational approach for the problem of extracting the roots of Arabic words based on the analysis of Arabic patterns and affix letters. This approach is purely a mathematical model which avoids having a list of the root and pattern of each word in the language. In order to achieve the main goal, the following tasks were accomplished:

1. A novel root stemming algorithm for Arabic language has been developed and tested.

2. A large database of Arabic roots was designed and collected from four famous Arabic dictionaries and stored in one dictionary.

3. A large corpus of Arabic words was designed, collected and annotated with the morphological features.

4. A comprehensive list of Arabic pattern forms were compiled and analysed. The list contains **1,893** patterns.

5. A comprehensive list of affix letters in Arabic language were compiled and analysed. This list contains **204** prefixes, **7** infixes and **193** suffixes.

6. A detailed analysis of the frequency of the affix letters in Arabic language was done.

7. A large list of Arabic stopwords was collected. The list contains **1,171** words.

8. A lot of experiments with different types of network topologies were run in order to determine the optimal size of the hidden neurons in the neural networks.

9. A detailed literature review of stemming algorithms and morphological analysers has been presented for Arabic and English languages.

8.5 Future Works

The MUAIDI-STEMMER is a first attempt of its kind for Arabic language. So, to attain better performance of the developed stemmer, we forward the following recommendations:

- Considering the analysis that has been done in this thesis as a springboard, one can extend the algorithm in order to cover other root types such as: **biliteral**, **quadriliteral**, and **pentaliteral**.

- The developed stemmer can also be used as a starting point for developing a stemmer for other Semitic languages such as Hebrew, Amharic, and Tigrinya[1].

- By implementing necessary elements, the stemmer can also be used as a component for developing other computational tools like morphological analyser, parser, spell checker, and part-of-speech tagger.

- Evaluating the MUAIDI-STEMMER using recall and precision evaluation method in information retrieval domains.

[1]The language used in Eritrea.

- Completing the compiled Arabic patterns in order to cover other pattern forms such as **quadriliteral** patterns.
- Completing the compiled Arabic stopwords.
- Investigating other neural network structures and learning algorithms in order to develop an optimal neural network structures.

Chapter 8

Bibliography

[1] Neural Network Toolbox [web document] available at:
`http://www.mathworks.com/access/helpdesk_r13/help/toolbox/nnet/tramnmx.html`
[accessed on 30/06/2007].

[2] Neural network toolbox Variable Learning Rate. [web document] available at:
`http://matlab.izmiran.ru/help/toolbox/nnet/backpr57.html`
[accessed on 12/11/2007].

[3] The Lovins stemming algorithm. [web document] available at:
`http://snowball.tartarus.org/algorithms/lovins/stemmer.html`
[accessed on 10/11/2006].

[4] ALS international and its affiliates. [web-document] available at:
`http://www.alsintl.com/languages/arabic.shtm`
[accessed on 21/09/2007].

[5] Abdelqadel Abdeljaleel. [المدارس المعجمية- دراسة في البنية التركيبية] (AlmdArs Alm'jmyt drAst fy Albnyt Altrkybyt, "The Dictionary Schools: A Study of their Structuers") [Arabic Book]. Dar Alsafa'. Amman, Jordan, 1997.

[6] Hani Abu-Salem. A Microcomputer Based Arabic Bibliographic Information Retrieval System with Relational Thesauri. PhD thesis, Department Of Computer Science, Illinois Institute Of Technology, Chicago, Il, 1992.

[7] Saleem Abuleil, Khalid Alsamara, and Martha Evens. Acquisition system for Arabic noun morphology. In Proceedings of The ACL-02 Workshop on Computational Approaches to Semitic Languages, pages 1–8. Association for Computational Linguistics, 2002.

[8] Saleem Abuleil and Martha Evens. Extracting an Arabic lexicon from Arabic newspaper text. Computers and the Humanities, 36(2):191–221, 2002.

[9] George W. Adamson and Jillian Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. Information Storage And Retrieval, 10(7-8):253–260, July - August 1974.

[10] Simon Ager. Arabic script. [web-document] available at:
`http://www.omniglot.com/writing/arabic.htm`
[accessed on 21/09/2007].

[11] B. H. Ahmad. Stemmer for a three letter unvocalized Arabic words. In The 1st Jordan International Conference on Computer Science & Engineering (JICCS'E04), pages 28–31, 2004.

[12] Mohamed Attia Ahmed. A large-scale computational processor of the Arabic morphology, andapplications. Master's thesis, Faculty Of Engineering, Cairo University, 2000.

[13] Shabbir Ahmed. Word stemming to enhance spam filtering. In CEAS 2004 - First Conference on Email and Anti-Spam, pages 30–31, July 2004.

[14] Al-Khalil Ibn Ahmed Al-Farahidy. [معجم العين](m'jmm Al'yn, "The Eye Dictionary") [Arabic Book]. Dar Ehya' Al-Turath Al-Arabi. Beirut - Lebanon, 1985.

[15] Sabah Al-Fedaghi and Fawaz Al-Anzi. A new algorithm to generate Arabic root-pattern forms. In Proceedings of the 11th National Computer and Exhibition, pages 391–400, March 1989.

[16] Sabah Al-Fedaghi and H. Al-Sadoun. Morphological compression of Arabic text. In Information Processing & Management, pages 303–316, 1990.

[17] Ahmad Al-Hamlawee. [شذا العرف في فن الصرف](ŝdA Al'rf fy fn AlSrf, "Essence of Morphology") [Arabic Book]. Dar Al-Kotob Al-Elmiyah. Beirut - Lebanon, 2000.

[18] Ibrahim A. Al-Kharashi and Martha Evens. Comparing words, stems, and roots as index terms in an Arabic information retrieval system. Journal Of The American Society For Information Science (JASIS), 45(8):548–560, 1994.

[19] Saleh Saleem Al-Phakhree. [تصريف الأسماء والمصادر والمشتقات](tSryf AlĀsmA' wAlmSAdr wAlmŝtqAt, "The Morphology of Nouns, Verbs and Derivatives") [Arabic Book]. Asmee for Publishing & Distribuation. Cairo, Eygpt, 1996.

[20] Abdoh Al-Rajehi. [في التطبيق النحوي والصرفي](fy AltTbyq AlnHwy wAl-Srfy, "In the Application of Syntax and Morphology") [Arabic Book]. Dar Al-Ma'refah Al-Jame'ayah. Al-Eskandariyah, Eygpt, 1992.

[21] Abu Baker Al-Razi. [معجم مختار الصحاح](m'jm mĤtAr AlSHAH, "Al-Sahah's Selection Dictionary") [Arabic Book]. Dar Al-Muhtaseb. Amman - Jordan, 1988.

[22] Hasan Al-Serhan and Aladdin Ayesh. An application of neural network for extracting Arabic word roots. WSEAS Transactions on Computers, 5(11):2623–2627, 2006.

[23] Hasan Al-Serhan and Aladdin Ayesh. A triliteral word roots extraction using neural network for Arabic. In IEEE International Conference on Computer Engineering and Systems (ICCES'06), pages 436–439, 2006.

[24] Riyad Al-Shalabi, Ghassan Kannan, Ahmad Ababneh, and Ahmad Al-Zubi. Experiments with the successor variety algorithm using the cut off and entropy methods. Information Technology Journal, 4(1):55–62, 2005.

[25] Riyad Al-Shalabi, Ghassan Kannan, and Hasan Al-Serhan. New approach for extracting Arabic roots. In ACIT '2003: Proceedings of the 2003 Arab Conference on Information Technology, volume 1, pages 42–59, 2003.

[26] Imad A. Al-Sughaiyer and Ibrahim Al-Kharashi. Firing policies for an Arabic rule-Based stemmer. In International Symposium On String Processing And Information Retrievalspire Lncs, volume 9, 2002.

[27] Imad A. Al-Sughaiyer and Ibrahim A. Al-kharashi. Rule parser for Arabic stemmer. In TSD '02: Proceedings Of The 5th International Conference On Text,speech And Dialogue, pages 11–18. Springer-verlag, 2002.

[28] Imad A. Al-Sughaiyer and Ibrahim A. Al-Kharashi. Arabic morphological analysis techniques: A comprehensive survey. Journal Of The American Society For Information Science And Technology, 55(3):189–213, 2004.

[29] Latifa Al-Sulaiti. [web document] available at:
`www.comp.leeds.ac.uk/eric/latifa/research.htm`
[accessed on 12/03/2006].

[30] Latifa Al-Sulaiti. Designing and developing a corpus of contemporary Arabic. MSc thesis school of computing, university of LEEDS, UK, 2004.

[31] Latifa Al-Sulaiti and Eric Atwel. Extending the corpus of contemporary Arabic. In Proceedings Of Corpus Linguistics Conference, 2005.

[32] Latifa Al-Sulaiti and Eric Atwell. The design of a corpus of contemporary Arabic. International Journal of Corpus Linguistics, 11(2):137–171, 2006.

[33] Murtada Al-Zubydee. [معجم تاج العروس](m'jm tAj Al'rws, "Bride's Crown Dictionary") [Arabic Book]. Dar Al-Kitab. Qairo - Egypt, 1992.

[34] N. Alemayehu and P. Willett. Stemming of Amharic words for information retrieval. Literary and Linguistic Computing, 17(1):1–17, 2002.

[35] N. Alemayehu and P. Willett. The effectiveness of stemming for information retrieval in amharic. Program, 37(4):254–259, 2003.

[36] N. Ali. Computers and Arabic Language. Al-Khat Publishing Press., 1988.

[37] Mohammed Aljlayl and Ophir Frieder. On Arabic search: Improving the retrieval effectiveness via a light stemming approach. In CIKM '02: Proceedings of the Eleventh International Conference on Information and Knowledge Management, pages 340–347. ACM, Sep. 2002.

[38] Fahed Ashour. Repetition in Mahmoud Darwish's Poetry. Arabic Establishment. Beirut, Lebanon, 2003.

[39] Fahed Ashour. Prevalence of grammatical rules throughout linguistic argumentative sources: A survey of some arabic grammatical vocables. Journal of Arabic Language, 2006.

[40] Fahed Ashour and H. Mahmoud. The Linguistic Structure of Eastern Dialects in Arabia: Emirates Dialect as a Model. Dar Al-Kitab. Irbid, Jordan, 2007.

[41] Jelita Asian, Hugh E. Williams, and S. M. M. Tahaghoghi. Stemming Indonesian. In Australian Computer Society Inc., pages 307–314, 2005.

[42] A. Asiri and Mohammad S. Khorsheed. Automatic processing of handwritten Arabic forms using neural networks. Transactions on Engineering, Computing and Technology, 7:1305–5313, August 2005.

[43] Nii O. Attoh-Okine. Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. Advances in Engineering Software, 30(4):291–302, 1999.

[44] Arafat Awajan. A rule-based morphological analyzer of Arabic words. In WSEAS Applied Mathematics 2003 and WSEAS IMCASS-ISA-SOSM, page 9, 2003.

[45] Kenneth R. Beesley. Romanization, transcription and transliteration - xer Arabic morphogical analysis and generation . [web document] available at: `http://www.xrce.xerox.com/competencies/content-analysis/arabic/info/romanization.html` [accessed on 19/12/2007].

[46] Kenneth R. Beesley. Arabic finite-state morphological analysis and generation. In COLING, pages 89–94, 1996.

[47] Kenneth R. Beesley. Consonant spreading in Arabic stems. In COLING-ACL, pages 117–123, 1998.

[48] Kenneth R. Beesley. Finite-state morphological analysis and generation of Arabic at xerox: Status and plans in 2001. In ACL Workshop on Arabic Language Processing: Status and Perspective, pages 1–8, 2001.

[49] Christopher M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press Inc., 1996.

[50] Tim Buckwalter. Arabic Morphological Analyzer Version 2.0, Linguistic [web document] available at:
`http://www.qamus.org/`
[accessed on 30/06/2007].

[51] Tim Buckwalter. Arabic transliteration encoding chart [web-document] available at:
`http://www.xrce.xerox.com/competencies/content-analysis/arabic/info/translit-chart.html`.
[accessed on 10/11/2007].

[52] J. Carlberger, H. Dalianis, M. Hassel, and O. Knutsson. Improving precision in information retrieval for swedish using stemming. In In The Proceedings Of Nodalida 01 - 13th Nordic Conference On Computationallinguistics, 2001.

[53] Violetta Cavalli-Sforza, Abdelhadi Soudi, and Teruko Mitamura. Arabic morphology generation using a concatenative strategy. In Morgan Kaufmann Publishers Inc., pages 86–93, 2000.

[54] Aitao Chen and Fredric C. Gey. Building an Arabic stemmer for information retrieval. In The Eleventh Text Retrieval Conference Gaithersburg (TREC 2002), number 19–22, 2002.

[55] Abdur Chowdhury, Mohammed Aljlayl, Eric C. Jensen, and David A. Grossman. IIT at TREC 2002 linear combinations based on document structure- and varied stemming for Arabic retrieval. In TREC, 2002.

[56] David Crystal. A Dictionary of Linguistics and Phonetics. Blackwell Publishers Ltd., 1997.

[57] A. Dahdah. [معجم قواعد اللغة العربية](m'jm qwA'd Allgt Al'rbyt,"The Dictionary of Arabic Language Grammar") [Arabic Book]. Lebanon Library. Beirut, Lebanon, 1985.

[58] Kareem Darwish. Building a shallow Arabic morphological analyzer in one day. In Proceedings of the ACL-02 workshop on Computational approaches to semitic languages, pages 1–8. Association for Computational Linguistics (Morristown, USA), 2002.

[59] Kareem Darwish and Douglas W. Oard. CLIR experiments at maryland for TREC-2002: Evidence combination for Arabic-english retrieval. Technical report, University of Maryland, CollegePark, February 2003.

[60] Dave and Erson. Artificial neural networks technology. Technical report, DACS - Data & Analysis Center For Software, August 1992.

[61] Mona Diab, Kadri Hacioglu, and Daniel Jurafsky. Automatic tagging of Arabic text: From raw text to base phrase chunks. In NAACL-HLT, 2004.

[62] F. Ccuna Ekmekccioglu, Michael F. Lynch, and Peter Willett. Stemming and N-Gram matching for term conflation in turkish texts. Information Research News, 2:2–6, 1996.

[63] T. A. El-Sadany and M. A. Hashish. An Arabic morphological system. IBM Systems Journal, 28(4):600–612, 1989.

[64] Mohamed Abdel Fattah, Fuji Ren, and Shingo Kuroiwa. Stemming to improve translation lexicon creation from bitexts. Information Processing and Management, 42(4):1003–1016, 2005.

[65] Laurene V. Fausett. Fundamentals of Neural Networks: Architectures, Algorithms and Applications. Prentice Hall, 1994.

[66] W. B. Frakes. Stemming algorithms. In Information Retrieval, Data Structures and Algorithms. Prentice Hall, 1992.

[67] William B. Frakes and Ricardo A. Baeza-Yates. Information Retrieval: Data Structures and Algorithms. Prentice Hall, 1992.

[68] M. Gheith and T. El-sadany. Arabic morphological analyzer on a personal computer. Technical report, Arabic Morphology Workshop. Stanford University, 1987.

[69] Abduelbaset Goweder and Anne De Roeck. Assessment of a significant Arabic corpus. In Arabic NLP Workshop at ACL/EACL 2001, 2001.

[70] M. Greengrass, Alexander M. Robertson, Reinhard Schinke, and Peter Willett. Processing morphological variants in searches of latin text. Information Research News, 2(1), 1996.

[71] Margaret A. Hafer and Stephen F. Weiss. Word segmentation by letter successor varieties. Information Storage and Retrieval, 10(11-12):371–385, November-December 1974.

[72] Martin T. Hagan, Howard D. Demuth, and Mark Beale. Neural Network Design. PWS Publishing Company- International Thomson Publishing, 1995.

[73] Patrick A. V. Hall. Design of information systems for Arabic. In Proceedings of the 2nd Conference of the European Cooperation on Informatics, pages 643–663. SPRINGER-VERLAG, 1978.

[74] Abdelmajid Ben Hamadou. A compression technique for Arabic dictionaries: The affix analysis. In Proceedings of the 11th Coference on Computational Linguistics, pages 286–288. Association for Computational Linguistics, 1986.

[75] Donna Harman. How effective is suffixing? Journal of the American Society for Information Science (JASIS), 42(1):7–15, 1991.

[76] Simon Haykin. Neural Networks: A Comprehensive Foundation. Prentice Hall Ptr, 1998.

[77] J A Haywood and H M Nahmad. A New Arabic Grammar of the Written Language. Lund Humphries Publishers Ltd, 2000.

[78] Abdelsalam Heddaya. Qalam: A convention formorphological Arabic-latin-Arabic transliteration. [web-document] available at: `http://langs.eserver.org/qalam.txt`. [accessed on 12/01/2008].

[79] John Hertz, Richard G. Palmer, and Ers Krogh. Introduction to the Theory of Neural Computation. Addison-wesley Publishing Company, Inc., 1991.

[80] Ismail Hmeidi, Ghassan Kanaan, and Martha Evens. Design and implementation of automatic indexing for information retrieval with Arabic documents. Journal Of The American Society For Information Science (JASIS), 48(10):867–881, 1997.

[81] Clive Holes. Modern Arabic: Structures, Functions And Varieties. Georgetown Classics In Arabic Language And Linguistics Series, Georgetown University Press, 2004.

[82] David A. Hull. Stemming algorithms: A case study for detailed evaluation. Journal Of The American Society For Information Science (JASIS), 47(1):70–84, 1996.

[83] Ibn-Mandhoor. [معجم لسان العرب](m'jm lsAn Al'rb,"The Tongue of the Arabs") [Arabic Book]. Dar Sader. Beirut - Lebanon, 1986.

[84] Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. Neural Networks, 1(4):295–307, 1988.

[85] Anil K. Jain, Jianchang Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. IEEE Computer, 29(3):31–44, March 1996.

[86] Jayathi Janakiraman and Vasant Honavar. Adaptive learning rate selection for backpropagation networks, September 1993.

[87] Jyh-Shing Roger Jang, Chuen-Tsai Sun, and Eiji Mizutani. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine intelligence. Prentice Hall, 1996.

[88] Daniel Jurafsky and James H. Martin. Speech and Language Processing: An introduction to Speech Recognition, Natural Language Processing, and Computational Linguistics, and Speech Recognition. Prentice-Hall, 2007.

[89] Ghassan Kanaan, Riyad Al-Shalabi, and Majdi Sawalha. Improving Arabic information retrieval systems using part of speechtagging. Information Technology Journal, 4(1):32–37, 2005.

[90] Ibrahim A. Al Kharashi and Imad A. Al Sughaiyer. Rule merging in a rule-based Arabic stemmer. In Proceedings Of The 19th International Conference On Computationallinguistics (COLING-02), pages 1–7. Association For Computational Linguistics, 2002.

[91] Ibrahim A. Al Kharashi and Imad A. Al Sughaiyer. Performance evaluation of an Arabic rule-based stemmer. In The 17th International Computer Science Conference, pages 405–415, April 2004.

[92] Shereen Khoja. APT: An automatic Arabic part-of-speech tagger. In Proceedings of the Student Workshop at NAACL, 2001.

[93] Shereen Khoja and R. Garside. Stemming Arabic text. Technical report, Computing Department, Lancaster University, 1999.

[94] Kimmo Koskenniemi. A general computational model for word-form recognition and production. In Proceedings of the 22nd Annual Meeting on Association for Computational Linguistics, pages 178–181. Association for Computational Linguistics, 1984.

[95] R. Krovetz. Viewing morphology as an inference process. In Proceedings Of The Sixteenth Annual International ACM SIGIR Conferenceon Research And Development In Information Retrieval, pages 191–203, 1993.

[96] Leah Larkey, Lisa Ballesteros, and Margaret E. Connell. Light stemming for Arabic information retrieval. Arabic Computational Morphology- Text, Speech and Language Technology Book Series - Springer, 38:221–243, 2007.

[97] Leah S. Larkey, Lisa Ballesteros, and Margaret E. Connell. Improving stemming for Arabic information retrieval: Light stemming and co-occurrence analysis. In SIGIR '02: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 275–282. ACM (New York, USA), 2002.

[98] Linguistic Data Consortium LDC. Buckwalter Morphological Analyzer Version 2.0. [web document] available at:
`http://www.ldc.upenn.edu/Catalog/`
[accessed on 10/04/2007].

[99] Martin Lennon, David S. Pierce, Brian D. Tarry, and Peter Willett. An evaluation of some conflation algorithms for information retrieval. Journal of Information Science, (3):177–183, 1981.

[100] Julie Beth Lovins. Development of a stemming algorithm. Mechanical Translation and Computational Linguistics, 11(1-2):22–31, 1968.

[101] Bassam Hammo Steven Lytinen, Saleem Abuleil, Steven Lytinen, and Martha Evens. Experimenting with a question answering system for the Arabic language. Computers and the Humanities, 38(1):397–415, 2004.

[102] W. S. Mcculloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin Of Mathematical Biophysics, 5, 1943.

[103] Anthony Mark McEnery. Computational Linguistics: A Handbook and Toolbox for Natural Language Processing. Sigma Press, Wilmslow, UK, 1992.

[104] Bassem Medawar. Classical Arabic transliteration (CAT) . [web-document] available at:
`http://almashriq.hiof.no/general/400/410/418/classical_arabic_transliteratio.`
[accessed on 21/01/2008].

[105] Mohanned Momani and Jamil Faraj. A novel algorithm to extract tri-literal Arabic roots. In IEEE/ACS International Conference On Computer Systems And Applications (AICCSA), pages 309–315. Ieee, May 2007.

[106] Haidar Moukdad. Stemming and root-Based approaches to the retrieval of Arabic documents on the web. Webology, 3(1), March 2006.

[107] Thabet Naglaa. Stemming the Qur'an. In COLING, pages 85–88, 2004.

[108] Masami Nakamura, Katsuteru Maruyama, Takeshi Kawabata, and Kiyohiroshikano. Neural network approach to word category prediction for english texts. In COLING, pages 213–218, 1990.

[109] Preslav Nakov. Building an inflectional stemmer for bulgarian. In COMPSYSTECH '03: Proceedings of the 4th International Conference Conference on Computer Systems and Technologies, pages 419–424. ACM, 2003.

[110] B.L. Narayan and Sankar K. Pal. Distribution based stemmer refinement. In First International Conference, PReMI (Pattern Recognition and Machine Intelligence), volume 3776, pages 672–677. Springer Berlin / Heidelberg, December 20-22 2005.

[111] Abdusalam Nwesri, S.M.M. Tahaghoghi, and Falk Scholer. Capturing out-of-vocabulary words in Arabic text. In Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, pages 258–266, July 2006.

[112] Carl F. J. Overhage. Project intrex: A general review. Information Storage and Retrieval, 10(5-6):157–188, 1974.

[113] Chris D. Paice. Another stemmer. SIGIR Forum, 24(3):56–61, 1990.

[114] Chris D. Paice. An evaluation method for stemming algorithms. In SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 42–50. SPRINGER-Verlag New York, INC., 1994.

[115] Chris D. Paice. Method for evaluation of stemming algorithms based on error counting. Journal of The American Society for Information Science (JASIS), 47(8):632–649, 1996.

[116] A.s. Pandya and R.b. Macy. Pattern Recognition With Neural Networks in C++. Crc Press, 1996.

[117] Ari Pirkola. Morphological typology of languages for IR. Journal of Documentation, 57(3):330–348, 2001.

[118] Mirko Popovic and Peter Willett. The effectiveness of stemming for natural language access to slovene textual data. Journal of The American Society for Information Science & Technology (JASIST), 43:384–390, 1992.

[119] Martin F. Porter. Snowball: A language for stemming algorithms [web document] available at: `http://snowball.tartarus.org/texts/introduction.html` [accessed on 12/01/2007].

[120] Martin F. Porter. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.

[121] Fagher Aldeen Qabaweh. [تصريف الأسماء والأفعال](tSryf AlsmA' wAlÃf'Al, "The Morphology of Nouns and Verbs") [Arabic Book]. Dar Al Maerf. Beirut - Lebanon, 1988.

[122] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Parallel Distributed Processing. Mit Press, 1986.

[123] D.e. Rumelhart, Bernard. Widrow, and M.a. Lehr. The basic ideas in neural networks. Communaction ACM, 37(3):87–92, 1994.

[124] Karin C. Ryding. A Reference Grammar of Modern Standard Arabic. Cambridge University Press, 2005.

[125] Gerard Salton. Automatic Text Processing: The Transformation, Analysis, and Retrieval of information By Computer. Addison-wesley Longman Publishing Co., Inc., 1989.

[126] Khaled Shaalan, Amin Allam, and AbdAllah Gomah. Towards automatic spell checking for Arabic. In Proceedings of the 4th Conference on Lan-

guage Engineering, Egyptian Society of Language Engineering (ELSE), pages 240–247, October 2003.

[127] Riyad Al Shalabi. A computational morphology system for Arabic. In COLING-ACL98, 1998.

[128] Riyad Al Shalabi. Pattern-based stemmer for finding Arabic roots. Information Technology Journal, 4(1):38–43, 2005.

[129] Jiri Sima. Back-propagation is not efficient. Neural Networks, 9(6):1017–1023, 1996.

[130] Jiri Sima. Introductions to neural networks. Technical report, Institute Of Computer Science, Academy Of Sciences Of The Czech Republic, August 1998.

[131] Gian Chand Sodhy. Prefix extraction of malay words using backpropagation neural network. In Proceedings Of The Fast Ieee Conference On Computer Science And Informationtechnology, pages 57–63, 1998.

[132] Robert R. Sokal and Peter H.A. Sneath. Principles of Numerical Taxonomy. W.H. Freeman, 1963.

[133] Abdelhadi Soudi, Antal Van Den Bosch, and Gunter Neumann. Arabic morphological generation and its impact on the quality of machine translation to Arabic. Arabic Computational Morphology; Text, Speech and Language Technology, 38:287–302, 2007.

[134] The Lovins Stemmer. [web document] available at:
`http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm`
[accessed on 10/11/2006].

[135] M. M. Syiam, Z. T. Fayed, and M. B. Habib. An intelligent system for Arabic text categorization. The International Journal Of Intelligent Computing And Information Sciences (IJICIS), 6(1):1–19, January 2006.

[136] Kazem Taghva, Rania Elkhoury, and Jelrey Coombs. Arabic stemming without a root dictionary. IEEE International Conference On Information Technology: Coding And Computing (ITCC'05), 1:152–157, 2005.

[137] Sock Yin Tai, Cheng Soon Ong, and Noor Aida Abullah. On designing an automated malaysian stemmer for the malay language (poster session). In IRAL '00: Proceedings of the Fifth International Workshop on Information Retrieval with Asian Languages, pages 207–208. ACM Press, 2000.

[138] Botrous Thalouth and Abdullah Al-Dannan. A comprehensive Arabic morphological analyser generator. Computers and the Arabic Language, pages 208–217, 1990.

[139] Vishwa Vinay. Automatic keyphrase assignment. MSc thesis, university college LONDON, 2003.

[140] Jinxi Xu, Alexander Fraser, and Ralph M. Weischedel. Empirical studies in strategies for Arabic retrieval. In SIGIR '02: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 269–274. ACM (New York, USA), 2002.

[141] Hilal Y. Automatic processing of Arabic language and application. In Proceedings of The Arabic Language Processing Using Computer Conference, pages 213–219, 1990.

[142] Ameel Yagoub. [المعاجم اللغوية العربية- بداءتها وتطورها] (Alm'Ajm Allgwyt Al'rbyt bdA'thA wtTwrhA, "Arabic Linguistic Dictionaries, their Starting and Development") [Arabic Book]. Dar Alelm Llmalayeen. Beirut, Lebanon, 1981.

Appendix

Appendix A

The Complete List of the Compiled Arabic Patterns

Table A.1: A List of Compiled Arabic Patterns for $patt_4$

Pattern	Pattern Form	Pattern	Pattern Form
افعل	$AC_1C_2C_3$	بفعل	$bC_1C_2C_3$
تفعل	$tC_1C_2C_3$	تفعل	$tC_1C_2C_3$
فاعل	$C_1AC_2C_3$	فعال	$C_1C_2AC_3$
فعلا	$C_1C_2C_3A$	فعلت	$C_1C_2C_3t$
فعلة	$C_1C_2C_3t$	فعلة	$C_1C_2C_3t$
فعلة	$C_1C_2C_3t$	فعلة	$C_1C_2C_3t$
فعلك	$C_1C_2C_3k$	فعلن	$C_1C_2C_3n$
فعله	$C_1C_2C_3h$	فعلو	$C_1C_2C_3w$
فعلى	$C_1C_2C_3A$	فعلي	$C_1C_2C_3y$
فعول	$C_1C_2wC_3$	فعيل	$C_1C_2yC_3$
ففعل	$fC_1C_2C_3$	فوعل	$C_1wC_2C_3$
فيعل	$C_1yC_2C_3$	كفعل	$kC_1C_2C_3$
لفعل	$lC_1C_2C_3$	مفعل	$mC_1C_2C_3$
نفعل	$nC_1C_2C_3$	وفعل	$wC_1C_2C_3$
يفعل	$yC_1C_2C_3$		

Table A.2: A List of Compiled Arabic Patterns for $patt_4$

Pattern	Pattern Form	Pattern	Pattern Form
اتفعل	$AtC_1C_2C_3$	افاعل	$AC_1AC_2C_3$
افتعل	$AC_1tC_2C_3$	افعال	$AC_1C_2AC_3$
افعلا	$AC_1C_2C_3A$	افعلة	$AC_1C_2C_3t$
افعلك	$AC_1C_2C_3k$	افعلن	$AC_1C_2C_3n$
افعله	$AC_1C_2C_3h$	افعلي	$AC_1C_2C_3y$
افعول	$AC_1C_2wC_3$	افعيل	$AC_1C_2yC_3$
الفعل	$AlC_1C_2C_3$	انفعل	$AnC_1C_2C_3$
بفعال	$bC_1C_2AC_3$	بفعلة	$bC_1C_2C_3t$
بفعله	$bC_1C_2C_3h$	بفعول	$bC_1C_2wC_3$
تتفعل	$ttC_1C_2C_3$	تفاعل	$tC_1AC_2C_3$
تفتعل	$tC_1tC_2C_3$	تفعال	$tC_1C_2AC_3$
تفعلا	$tC_1C_2C_3A$	تفعلة	$tC_1C_2C_3t$
تفعلك	$tC_1C_2C_3k$	تفعلن	$tC_1C_2C_3n$
تفعله	$tC_1C_2C_3h$	تفعلي	$tC_1C_2C_3y$
تفعول	$tC_1C_2wC_3$	تفعيل	$tC_1C_2yC_3$
تمفعل	$tmC_1C_2C_3$	تنفعل	$tnC_1C_2C_3$
سافعل	$sAC_1C_2C_3$	ستفعل	$stC_1C_2C_3$
سنفعل	$snC_1C_2C_3$	سيفعل	$syC_1C_2C_3$
فاعال	$C_1AC_2AC_3$	فاعلا	$C_1AC_2C_3A$
فاعلة	$C_1AC_2C_3t$	فاعلن	$C_1AC_2C_3n$
فاعلو	$C_1AC_2C_3w$	فاعلي	$C_1AC_2C_3y$
فاعول	$C_1AC_2wC_3$	فبفعل	$fbC_1C_2C_3$
فعائل	$C_1C_2AAC_3$	فعالا	$C_1C_2AC_3A$
فعالة	$C_1C_2AC_3t$	فعالن	$C_1C_2AC_3n$
فعاله	$C_1C_2AC_3h$	فعالى	$C_1C_2AC_3A$
فعالي	$C_1C_2AC_3y$	فعاول	$C_1C_2AwC_3$
فعايل	$C_1C_2AyC_3$	فعلاء	$C_1C_2C_3A'$
فعلاة	$C_1C_2C_3At$	فعلات	$C_1C_2C_3At$
فعلاك	$C_1C_2C_3Ak$	فعلان	$C_1C_2C_3An$
فعلاه	$C_1C_2C_3Ah$	فعلتا	$C_1C_2C_3tA$
فعلتك	$C_1C_2C_3tk$	فعلتم	$C_1C_2C_3tm$
فعلتن	$C_1C_2C_3tn$	فعلته	$C_1C_2C_3th$
فعلكم	$C_1C_2C_3km$	فعلكن	$C_1C_2C_3kn$
فعلنا	$C_1C_2C_3nA$	فعلنك	$C_1C_2C_3nk$
فعلنه	$C_1C_2C_3nh$	فعلني	$C_1C_2C_3ny$
فعلها	$C_1C_2C_3hA$	فعلهم	$C_1C_2C_3hm$
فعلهن	$C_1C_2C_3hn$	فعلوا	$C_1C_2C_3wA$
فعلوك	$C_1C_2C_3wk$	فعلوه	$C_1C_2C_3wh$
فعليا	$C_1C_2C_3yA$	فعلية	$C_1C_2C_3yt$
فعلين	$C_1C_2C_3yn$	فعوال	$C_1C_2wAC_3$
فعولا	$C_1C_2wC_3A$	فعولة	$C_1C_2wC_3t$
فعولى	$C_1C_2wC_3A$	فعيال	$C_1C_2yAC_3$
فعيلة	$C_1C_2yC_3t$	فعيله	$C_1C_2yC_3h$
فعيلي	$C_1C_2yC_3y$	ففعلة	$fC_1C_2C_3t$
ففعله	$fC_1C_2C_3h$	فواعل	$C_1wAC_2C_3$

table continued on next page

Table A.2 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
فوعلا	$C_1wC_2C_3A$	فوعلة	$C_1wC_2C_3t$
فياعل	$C_1yAC_2C_3$	فيعال	$C_1yC_2AC_3$
فيعول	$C_1yC_2wC_3$	كفاعل	$kC_1AC_2C_3$
لتفعل	$ltC_1C_2C_3$	لفعلة	$lC_1C_2C_3t$
لفعله	$lC_1C_2C_3h$	للفعل	$llC_1C_2C_3$
متفعل	$mtC_1C_2C_3$	مفاعل	$mC_1AC_2C_3$
مفاعل	$mC_1AC_2C_3$	مفتعل	$mC_1tC_2C_3$
مفعال	$mC_1C_2AC_3$	مفعلا	$mC_1C_2C_3A$
مفعلة	$mC_1C_2C_3t$	مفعلو	$mC_1C_2C_3w$
مفعلي	$mC_1C_2C_3y$	مفعول	$mC_1C_2wC_3$
مفعيل	$mC_1C_2yC_3$	مفيعل	$mC_1yC_2C_3$
منفعل	$mnC_1C_2C_3$	نتفعل	$ntC_1C_2C_3$
نفاعل	$nC_1AC_2C_3$	نفتعل	$nC_1tC_2C_3$
نفعلك	$nC_1C_2C_3k$	نفعله	$nC_1C_2C_3h$
نفعول	$nC_1C_2wC_3$	نفوعل	$nC_1wC_2C_3$
ننفعل	$nnC_1C_2C_3$	وتفعل	$wtC_1C_2C_3$
وفعلة	$wC_1C_2C_3t$	وفعله	$wC_1C_2C_3h$
وفعول	$wC_1C_2wC_3$	وفعيل	$wC_1C_2yC_3$
ويفعل	$wyC_1C_2C_3$	يتفعل	$ytC_1C_2C_3$
يفاعل	$yC_1AC_2C_3$	يفتعل	$yC_1tC_2C_3$
يفعلا	$yC_1C_2C_3A$	يفعلك	$yC_1C_2C_3k$
يفعلن	$yC_1C_2C_3n$	يفعله	$yC_1C_2C_3h$
يفوعل	$yC_1wC_2C_3$	ينفعل	$ynC_1C_2C_3$

Table A.3: A List of Compiled Arabic Patterns for $patt_6$

Pattern	Pattern Form	Pattern	Pattern Form
اتفاعل	$AtC_1AC_2C_3$	استفعل	$AstC_1C_2C_3$
افتعال	$AC_1tC_2AC_3$	افتعلت	$AC_1tC_2C_3t$
افتوعل	$AC_1twC_2C_3$	افعالا	$AC_1C_2AC_3A$
افعاله	$AC_1C_2AC_3h$	افعالي	$AC_1C_2AC_3y$
افعلاء	$AC_1C_2C_3A'$	افعلاه	$AC_1C_2C_3Ah$
افعلتا	$AC_1C_2C_3tA$	افعلتم	$AC_1C_2C_3tm$
افعلتن	$AC_1C_2C_3tn$	افعلكم	$AC_1C_2C_3km$
افعلكن	$AC_1C_2C_3kn$	افعلنا	$AC_1C_2C_3nA$
افعلنه	$AC_1C_2C_3nh$	افعلني	$AC_1C_2C_3ny$
افعلها	$AC_1C_2C_3hA$	افعلهم	$AC_1C_2C_3hm$
افعلهن	$AC_1C_2C_3hn$	افعلوا	$AC_1C_2C_3wA$
افعلوه	$AC_1C_2C_3wh$	افعليه	$AC_1C_2C_3yh$
افنفعل	$AfnC_1C_2C_3$	التفعل	$AltC_1C_2C_3$
الفاعل	$AlC_1AC_2C_3$	الفعال	$AlC_1C_2AC_3$
الفعلة	$AlC_1C_2C_3t$	الفعلى	$AlC_1C_2C_3A$
الفعلي	$AlC_1C_2C_3y$	الفعول	$AlC_1C_2wC_3$

table continued on next page

Table A.3 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
الفعيل	$AlC_1C_2yC_3$	الفيعل	$AlC_1yC_2C_3$
المفعل	$AlmC_1C_2C_3$	انفعال	$AnC_1C_2AC_3$
انفعلا	$AnC_1C_2C_3A$	انفعلت	$AnC_1C_2C_3t$
انفعلن	$AnC_1C_2C_3n$	انفعلي	$AnC_1C_2C_3y$
بالفعل	$bAlC_1C_2C_3$	بتفعيل	$btC_1C_2yC_3$
بفعاله	$bC_1C_2AC_3h$	بفعوال	$bC_1C_2wAC_3$
بفعولة	$bC_1C_2wC_3t$	بفعيلة	$bC_1C_2yC_3t$
بمفتعل	$bmC_1tC_2C_3$	بمفعله	$bmC_1C_2C_3h$
تتفاعل	$ttC_1AC_2C_3$	تتفعلا	$ttC_1C_2C_3A$
تتفعلن	$ttC_1C_2C_3n$	تتفعلي	$ttC_1C_2C_3y$
تستفعل	$tstC_1C_2C_3$	تفاعلا	$tC_1AC_2C_3A$
تفاعلت	$tC_1AC_2C_3t$	تفاعلن	$tC_1AC_2C_3n$
تفاعلي	$tC_1AC_2C_3y$	تفاعيل	$tC_1AC_2yC_3$
تفعلات	$tC_1C_2C_3At$	تفعلان	$tC_1C_2C_3An$
تفعلتا	$tC_1C_2C_3tA$	تفعلتم	$tC_1C_2C_3tm$
تفعلتن	$tC_1C_2C_3tn$	تفعلكم	$tC_1C_2C_3km$
تفعلكن	$tC_1C_2C_3kn$	تفعلنا	$tC_1C_2C_3nA$
تفعلنه	$tC_1C_2C_3nh$	تفعلني	$tC_1C_2C_3ny$
تفعلها	$tC_1C_2C_3hA$	تفعلهم	$tC_1C_2C_3hm$
تفعلهن	$tC_1C_2C_3hn$	تفعلوا	$tC_1C_2C_3wA$
تفعلون	$tC_1C_2C_3wn$	تفعلين	$tC_1C_2C_3yn$
تفعيله	$tC_1C_2yC_3h$	تفعيلي	$tC_1C_2yC_3y$
تنفعلا	$tnC_1C_2C_3A$	تنفعلن	$tnC_1C_2C_3n$
تنفعلي	$tnC_1C_2C_3y$	سافعله	$sAC_1C_2C_3h$
ستفعله	$stC_1C_2C_3h$	سنفعله	$snC_1C_2C_3h$
فاعلات	$C_1AC_2C_3At$	فاعلان	$C_1AC_2C_3An$
فاعلتا	$C_1AC_2C_3tA$	فاعلتم	$C_1AC_2C_3tm$
فاعلتن	$C_1AC_2C_3tn$	فاعلتي	$C_1AC_2C_3ty$
فاعلنا	$C_1AC_2C_3nA$	فاعلوا	$C_1AC_2C_3wA$
فاعلون	$C_1AC_2C_3wn$	فاعلية	$C_1AC_2C_3yt$
فاعلين	$C_1AC_2C_3yn$	فتتفعل	$fttC_1C_2C_3$
فعلنها	$C_1C_2AC_3hA$	فعالات	$C_1C_2AC_3At$
فعالتي	$C_1C_2AC_3ty$	فعالنه	$C_1C_2AC_3nh$
فعالها	$C_1C_2AC_3hA$	فعالية	$C_1C_2AC_3yt$
فعلاتي	$C_1C_2C_3Aty$	فعلاكم	$C_1C_2C_3Akm$
فعلاكن	$C_1C_2C_3Akn$	فعلانا	$C_1C_2C_3AnA$
فعلاني	$C_1C_2C_3Any$	فعلاها	$C_1C_2C_3AhA$
فعلاهم	$C_1C_2C_3Ahm$	فعلاهن	$C_1C_2C_3Ahn$
فعلتاك	$C_1C_2C_3tAk$	فعلتاه	$C_1C_2C_3tAh$
فعلتكم	$C_1C_2C_3tkm$	فعلتكن	$C_1C_2C_3tkn$
فعلتما	$C_1C_2C_3tmA$	فعلتنا	$C_1C_2C_3tnA$
فعلتني	$C_1C_2C_3tny$	فعلتها	$C_1C_2C_3thA$
فعلتهم	$C_1C_2C_3thm$	فعلتهن	$C_1C_2C_3thn$
فعلكما	$C_1C_2C_3kmA$	فعلناك	$C_1C_2C_3nAk$
فعلناه	$C_1C_2C_3nAh$	فعلنكم	$C_1C_2C_3nkm$

table continued on next page

Table A.3 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
فعلنكن	$C_1C_2C_3nkn$	فعلننا	$C_1C_2C_3nnA$
فعلنني	$C_1C_2C_3nny$	فعلنها	$C_1C_2C_3nhA$
فعلنهم	$C_1C_2C_3nhm$	فعلنهن	$C_1C_2C_3nhn$
فعلهما	$C_1C_2C_3hmA$	فعلوكم	$C_1C_2C_3wkm$
فعلوكن	$C_1C_2C_3wkn$	فعلونا	$C_1C_2C_3wnA$
فعلوني	$C_1C_2C_3wny$	فعلوها	$C_1C_2C_3whA$
فعلوهم	$C_1C_2C_3whm$	فعلوهن	$C_1C_2C_3whn$
فعليات	$C_1C_2C_3yAt$	فعولها	$C_1C_2wC_3hA$
فعولهم	$C_1C_2wC_3hm$	فعيلات	$C_1C_2yC_3At$
فعيلان	$C_1C_2yC_3An$	فعيلون	$C_1C_2yC_3wn$
فواعيل	$C_1wAC_2yC_3$	فوعلتا	$C_1wC_2C_3tA$
فوعلتم	$C_1wC_2C_3tm$	فوعلتن	$C_1wC_2C_3tn$
فوعلنا	$C_1wC_2C_3nA$	فوعلوا	$C_1wC_2C_3wA$
كالفعل	$kAlC_1C_2C_3$	لفعيلة	$lC_1C_2yC_3t$
للتفعل	$lltC_1C_2C_3$	للفعال	$llC_1C_2AC_3$
ليفعلن	$lyC_1C_2C_3n$	متفاعل	$mtC_1AC_2C_3$
مستفعل	$mstC_1C_2C_3$	مفاعلة	$mC_1AC_2C_3t$
مفاعيل	$mC_1AC_2yC_3$	مفعلات	$mC_1C_2C_3At$
مفعلان	$mC_1C_2C_3An$	مفعلتي	$mC_1C_2C_3ty$
مفعلوا	$mC_1C_2C_3wA$	مفعلون	$mC_1C_2C_3wn$
مفعلين	$mC_1C_2C_3yn$	مفعوله	$mC_1C_2wC_3h$
نتفاعل	$ntC_1AC_2C_3$	نستفعل	$nstC_1C_2C_3$
نفعلكم	$nC_1C_2C_3km$	نفعلكن	$nC_1C_2C_3kn$
نفعلها	$nC_1C_2C_3hA$	نفعلهم	$nC_1C_2C_3hm$
نفعلهن	$nC_1C_2C_3hn$	وافعال	$wAC_1C_2AC_3$
والفعل	$wAlC_1C_2C_3$	وتفعيل	$wtC_1C_2yC_3$
وسافعل	$wsAC_1C_2C_3$	وستفعل	$wstC_1C_2C_3$
وسنفعل	$wsnC_1C_2C_3$	وفاعلة	$wC_1AC_2C_3t$
وفعالة	$wC_1C_2AC_3t$	وفعلاء	$wC_1C_2C_3A'$
وفعلها	$wC_1C_2C_3hA$	وفعلهم	$wC_1C_2C_3hm$
ومفاعل	$wmC_1AC_2C_3$	يتفاعل	$ytC_1AC_2C_3$
يتفعلا	$ytC_1C_2C_3A$	يتفعلن	$ytC_1C_2C_3n$
يستفعل	$ystC_1C_2C_3$	يفاعلا	$yC_1AC_2C_3A$
يفاعلن	$yC_1AC_2C_3n$	يفعلان	$yC_1C_2C_3An$
يفعلكم	$yC_1C_2C_3km$	يفعلكن	$yC_1C_2C_3kn$
يفعلنا	$yC_1C_2C_3nA$	يفعلنه	$yC_1C_2C_3nh$
يفعلني	$yC_1C_2C_3ny$	يفعلها	$yC_1C_2C_3hA$
يفعلهم	$yC_1C_2C_3hm$	يفعلهن	$yC_1C_2C_3hn$
يفعلوا	$yC_1C_2C_3wA$	يفعلون	$yC_1C_2C_3wn$
ينفعلا	$ynC_1C_2C_3A$	ينفعلن	$ynC_1C_2C_3n$

Table A.4: A List of Compiled Arabic Patterns for $patt_7$

Pattern	Pattern Form	Pattern	Pattern Form
اتستفعل	$AtstC_1C_2C_3$	استفعال	$AstC_1C_2AC_3$
استفعلا	$AstC_1C_2C_3A$	استفعلت	$AstC_1C_2C_3t$
استفعلك	$AstC_1C_2C_3k$	استفعلن	$AstC_1C_2C_3n$
افتعاله	$AC_1tC_2AC_3h$	افتعالي	$AC_1tC_2AC_3y$
افعالات	$AC_1C_2AC_3At$	افعالها	$AC_1C_2AC_3hA$
افعالية	$AC_1C_2AC_3yt$	افعلاها	$AC_1C_2C_3AhA$
افعلاهم	$AC_1C_2C_3Ahm$	افعلاهن	$AC_1C_2C_3Ahn$
افعلتما	$AC_1C_2C_3tmA$	افعلكما	$AC_1C_2C_3kmA$
افعلتها	$AC_1C_2C_3nhA$	افعلنهم	$AC_1C_2C_3nhm$
افعلنهن	$AC_1C_2C_3nhn$	افعلهما	$AC_1C_2C_3hmA$
افعلوها	$AC_1C_2C_3whA$	افعلوهم	$AC_1C_2C_3whm$
افعلوهن	$AC_1C_2C_3whn$	افعليها	$AC_1C_2C_3yhA$
افعليهم	$AC_1C_2C_3yhm$	افعليهن	$AC_1C_2C_3yhn$
الافعال	$AlAC_1C_2AC_3$	التفاعل	$AltC_1AC_2C_3$
التفعيل	$AltC_1C_2yC_3$	الفاعلة	$AlC_1AC_2C_3t$
الفاعلي	$AlC_1AC_2C_3y$	الفعئلي	$AlC_1C_2AC_3y$
الفعائل	$AlC_1C_2AAC_3$	الفعالة	$AlC_1C_2AC_3t$
الفعالي	$AlC_1C_2AC_3y$	الفعلاء	$AlC_1C_2C_3A'$
الفعلات	$AlC_1C_2C_3At$	الفعلان	$AlC_1C_2C_3An$
الفعلية	$AlC_1C_2C_3yt$	الفعلين	$AlC_1C_2C_3yn$
الفعولي	$AlC_1C_2wC_3y$	الفعيلة	$AlC_1C_2yC_3t$
الفوعلة	$AlC_1wC_2C_3t$	المتفعل	$AlmtC_1C_2C_3$
المفاعل	$AlmC_1AC_2C_3$	المفتعل	$AlmC_1tC_2C_3$
المفعال	$AlmC_1C_2AC_3$	المفعله	$AlmC_1C_2C_3h$
المفعلة	$AlmC_1C_2C_3t$	المفعول	$AlmC_1C_2wC_3$
المفعلي	$AlmC_1C_2C_3y$	انفعلتم	$AnC_1C_2C_3tm$
انفعلتا	$AnC_1C_2C_3tA$	انفعلنا	$AnC_1C_2C_3nA$
انفعلتن	$AnC_1C_2C_3tn$	ايفعلني	$AyC_1C_2C_3ny$
انفعلوا	$AnC_1C_2C_3wA$	بالفعال	$bAlC_1C_2AC_3$
بافتعال	$bAC_1tC_2AC_3$	بالمفعل	$bAlmC_1C_2C_3$
بالفعيل	$bAlC_1C_2yC_3$	بمفعلتي	$bmC_1C_2C_3ty$
بمفاعلة	$bmC_1AC_2C_3t$	تتفاعلن	$ttC_1AC_2C_3n$
تتفاعلا	$ttC_1AC_2C_3A$	تتفعلوا	$ttC_1C_2C_3wA$
تتفاعلي	$ttC_1AC_2C_3y$	تستفعلن	$tstC_1C_2C_3n$
تستفعلا	$tstC_1C_2C_3A$	تفاعلان	$tC_1AC_2C_3An$
تستفعلي	$tstC_1C_2C_3y$	تفاعلتم	$tC_1AC_2C_3tm$
تفاعلتا	$tC_1AC_2C_3tA$	تفاعلنا	$tC_1AC_2C_3nA$
تفاعلتن	$tC_1AC_2C_3tn$	تفاعلون	$tC_1AC_2C_3wn$
تفاعلوا	$tC_1AC_2C_3wA$	تفعلانه	$tC_1C_2C_3Anh$
تفاعلين	$tC_1AC_2C_3yn$	تفعلكما	$tC_1C_2C_3kmA$
تفعلتما	$tC_1C_2C_3tmA$	تفعلنهم	$tC_1C_2C_3nhm$
تفعلنها	$tC_1C_2C_3nhA$	تفعلهما	$tC_1C_2C_3hmA$
تفعلنهن	$tC_1C_2C_3nhn$	تفعيلها	$tC_1C_2yC_3hA$
تفعلونه	$tC_1C_2C_3wnh$	تنفعلوا	$tnC_1C_2C_3wA$
تنفعلان	$tnC_1C_2C_3An$		

table continued on next page

Table A.4 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
تنفعلون	$tnC_1C_2C_3wn$	تنفعلين	$tnC_1C_2C_3yn$
سافعلهم	$sAC_1C_2C_3hm$	فاعلاتي	$C_1AC_2C_3Aty$
فاعلتان	$C_1AC_2C_3tAn$	فاعلتما	$C_1AC_2C_3tmA$
فاعلتنا	$C_1AC_2C_3tnA$	فاعلتين	$C_1AC_2C_3tyn$
فافعلنا	$fAC_1C_2C_3nA$	فعالاتي	$C_1C_2AC_3Aty$
فعلاتهم	$C_1C_2C_3Athm$	فعلاكما	$C_1C_2C_3AkmA$
فعلاهما	$C_1C_2C_3AhmA$	فعلتاكم	$C_1C_2C_3tAkm$
فعلتاكن	$C_1C_2C_3tAkn$	فعلتانا	$C_1C_2C_3tAnA$
فعلتاني	$C_1C_2C_3tAny$	فعلتاها	$C_1C_2C_3tAhA$
فعلتاهم	$C_1C_2C_3tAhm$	فعلتاهن	$C_1C_2C_3tAhn$
فعلتكما	$C_1C_2C_3tkmA$	فعلتموه	$C_1C_2C_3tmwh$
فعلتهما	$C_1C_2C_3thmA$	فعلتوها	$C_1C_2C_3twhA$
فعلتوهم	$C_1C_2C_3twhm$	فعلناكم	$C_1C_2C_3nAkm$
فعلناكن	$C_1C_2C_3nAkn$	فعلنانا	$C_1C_2C_3nAnA$
فعلناني	$C_1C_2C_3nAny$	فعلناها	$C_1C_2C_3nAhA$
فعلناهم	$C_1C_2C_3nAhm$	فعلناهن	$C_1C_2C_3nAhn$
فعلنكما	$C_1C_2C_3nkmA$	فعلنهما	$C_1C_2C_3nhmA$
فعلوكما	$C_1C_2C_3wkmA$	فعلوهما	$C_1C_2C_3whmA$
فعليتها	$C_1C_2C_3ythA$	فعوليات	$C_1C_2wC_3yAt$
فعيلتان	$C_1C_2yC_3tAn$	فعيلتين	$C_1C_2yC_3tyn$
فوعلتما	$C_1wC_2C_3tmA$	كالمفعل	$kAlmC_1C_2C_3$
كمفعلتي	$kmC_1C_2C_3ty$	لتفعلان	$ltC_1C_2C_3An$
لفعالته	$lC_1C_2AC_3th$	للتفعيل	$lltC_1C_2yC_3$
لمفاعلة	$lmC_1AC_2C_3t$	مستفعلة	$mstC_1C_2C_3t$
مفاعلات	$mC_1AC_2C_3At$	مفاعلته	$mC_1AC_2C_3th$
مفاعلهم	$mC_1AC_2C_3hm$	مفعالان	$mC_1C_2AC_3An$
مفعلاتي	$mC_1C_2C_3Aty$	مفعلاني	$mC_1C_2C_3Any$
مفعلتان	$mC_1C_2C_3tAn$	مفعولات	$mC_1C_2wC_3At$
مفعولون	$mC_1C_2wC_3wn$	نفعلكما	$nC_1C_2C_3kmA$
نفعلهما	$nC_1C_2C_3hmA$	واستفعل	$wAstC_1C_2C_3$
وافتعال	$wAC_1tC_2AC_3$	والفاعل	$wAlC_1AC_2C_3$
والفعال	$wAlC_1C_2AC_3$	والفعول	$wAlC_1C_2wC_3$
والفعيل	$wAlC_1C_2yC_3$	والمفعل	$wAlmC_1C_2C_3$
وانفعال	$wAnC_1C_2AC_3$	وتتفاعل	$wttC_1AC_2C_3$
وتفاعيل	$wtC_1AC_2yC_3$	وفعولهم	$wC_1C_2wC_3hm$
يتفاعلا	$ytC_1AC_2C_3A$	يتفاعلن	$ytC_1AC_2C_3n$
يتفعلوا	$ytC_1C_2C_3wA$	يتفعلون	$ytC_1C_2C_3wn$
يستفعلا	$ystC_1C_2C_3A$	يفاعلان	$yC_1AC_2C_3An$
يفاعلوا	$yC_1AC_2C_3wA$	يفاعلون	$yC_1AC_2C_3wn$
يفتعلون	$yC_1tC_2C_3wn$	يفعلانه	$yC_1C_2C_3Anh$
يفعلكما	$yC_1C_2C_3kmA$	يفعلنها	$yC_1C_2C_3nhA$
يفعلنهم	$yC_1C_2C_3nhm$	يفعلنهن	$yC_1C_2C_3nhn$
يفعلهما	$yC_1C_2C_3hmA$	يفعلونه	$yC_1C_2C_3wnh$
ينفعلان	$ynC_1C_2C_3An$	ينفعلوا	$ynC_1C_2C_3wA$
ينفعلون	$ynC_1C_2C_3wn$		

Table A.5: A List of Compiled Arabic Patterns for $patt_8$

Pattern	Pattern Form	Pattern	Pattern Form
اتفاعلها	$AtC_1AC_2C_3hA$	اتفتعلان	$AtC_1tC_2C_3An$
استفعالا	$AstC_1C_2AC_3A$	استفعاله	$AstC_1C_2AC_3h$
استفعالي	$AstC_1C_2AC_3y$	استفعلات	$AstC_1C_2C_3At$
استفعلتا	$AstC_1C_2C_3tA$	استفعلتم	$AstC_1C_2C_3tm$
استفعلتن	$AstC_1C_2C_3tn$	استفعلته	$AstC_1C_2C_3th$
استفعلنا	$AstC_1C_2C_3nA$	استفعلني	$AstC_1C_2C_3ny$
استفعلها	$AstC_1C_2C_3hA$	استفعلهم	$AstC_1C_2C_3hm$
استفعلوا	$AstC_1C_2C_3wA$	استفعلوه	$AstC_1C_2C_3wh$
استفعلوه	$AstC_1C_2C_3wh$	استفعليا	$AstC_1C_2C_3yA$
افاعلكما	$AC_1AC_2C_3kmA$	افاعيلها	$AC_1AC_2yC_3hA$
افاعيلهم	$AC_1AC_2yC_3hm$	افتعالات	$AC_1tC_2AC_3At$
افتعالان	$AC_1tC_2AC_3An$	افتعالنا	$AC_1tC_2AC_3nA$
افتعالها	$AC_1tC_2AC_3hA$	افتعالهم	$AC_1tC_2AC_3hm$
افتعاليا	$AC_1tC_2AC_3yA$	افتعالية	$AC_1tC_2AC_3yt$
افتعالين	$AC_1tC_2AC_3ym$	افتعلتني	$AC_1tC_2C_3tny$
افتعلتها	$AC_1tC_2C_3thA$	افتعلتهم	$AC_1tC_2C_3thm$
افتعلونا	$AC_1tC_2C_3wnA$	افتعلوني	$AC_1tC_2C_3wny$
افتعلوها	$AC_1tC_2C_3whA$	افعالاته	$AC_1C_2AC_3Ath$
افعالاتي	$AC_1C_2AC_3Aty$	افعالهما	$AC_1C_2AC_3hmA$
افعاليات	$AC_1C_2AC_3yAt$	افعاليون	$AC_1C_2AC_3ywn$
افعاليين	$AC_1C_2AC_3yyn$	افعلاتهم	$AC_1C_2C_3Athm$
افعلاهما	$AC_1C_2C_3AhmA$	افعلنهما	$AC_1C_2C_3nhmA$
افعلوهما	$AC_1C_2C_3whmA$	افعليهما	$AC_1C_2C_3yhmA$
الافاعيل	$AlAC_1AC_2yC_3$	الافتعال	$AlAC_1tC_2AC_3$
الافعالي	$AlAC_1C_2AC_3y$	الافعلاء	$AlAC_1C_2C_3A'$
الافعلات	$AlAC_1C_2C_3At$	الافعلية	$AlAC_1C_2C_3yt$
الافعيال	$AlAC_1C_2yAC_3$	الافعيلي	$AlAC_1C_2yC_3y$
الانفعال	$AlAnC_1C_2AC_3$	التفعلات	$AltC_1C_2C_3At$
التفعيلي	$AltC_1C_2yC_3y$	الفاعلات	$AlC_1AC_2C_3At$
الفاعلان	$AlC_1AC_2C_3An$	الفاعلية	$AlC_1AC_2C_3yt$
الفاعلين	$AlC_1AC_2C_3ym$	الفاواعل	$AlC_1AwAC_2C_3$
الفعالات	$AlC_1C_2AC_3At$	الفعالون	$AlC_1C_2AC_3wn$
الفعالية	$AlC_1C_2AC_3yt$	الفعلتين	$AlC_1C_2C_3tym$
الفعليات	$AlC_1C_2C_3yAt$	الفعليني	$AlC_1C_2C_3yny$
الفعليين	$AlC_1C_2C_3yym$	الفعولات	$AlC_1C_2wC_3At$
الفعولية	$AlC_1C_2wC_3yt$	الفعيلات	$AlC_1C_2yC_3At$
الفعيلية	$AlC_1C_2yC_3yt$	الفواعيل	$AlC_1wAC_2yC_3$
المتفعلة	$AlmtC_1C_2C_3t$	المفاعلة	$AlmC_1AC_2C_3t$
المفاعيل	$AlmC_1AC_2yC_3$	المفعلات	$AlmC_1C_2C_3At$
المفعلان	$AlmC_1C_2C_3An$	المفعلون	$AlmC_1C_2C_3wn$
المفعلية	$AlmC_1C_2C_3yt$	المفعلين	$AlmC_1C_2C_3yn$
المفعولة	$AlmC_1C_2wC_3t$	المنفعلة	$AlmnC_1C_2C_3t$
انفعلتما	$AnC_1C_2C_3tmA$	باستفعال	$bAstC_1C_2AC_3$
بافتعالة	$bAC_1tC_2AC_3t$	بافتعاله	$bAC_1tC_2AC_3h$
بافتعالي	$bAC_1tC_2AC_3y$	بافعالات	$bAC_1C_2AC_3At$

table continued on next page

Table A.5 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
بافعالنا	$bAC_1C_2AC_3nA$	بافعالها	$bAC_1C_2AC_3hA$
بافعالهم	$bAC_1C_2AC_3hm$	بافعالهن	$bAC_1C_2AC_3hn$
بافعلتهم	$bAC_1C_2C_3thm$	بالافعال	$bAlAC_1C_2AC_3$
بالفاعول	$bAlC_1AC_2wC_3$	بالفعائل	$bAlC_1C_2AAC_3$
بالفعالة	$bAlC_1C_2AC_3t$	بالفعلية	$bAlC_1C_2C_3yt$
بالفوعلة	$bAlC_1wC_2C_3t$	بالمفاعل	$bAlmC_1AC_2C_3$
بالمفعلة	$bAlmC_1C_2C_3t$	بتفعلتها	$btC_1C_2C_3thA$
بتفعيلات	$btC_1C_2yC_3At$	بتفعيلها	$btC_1C_2yC_3hA$
بفاعلهما	$bC_1AC_2C_3hmA$	بفعالاته	$bC_1C_2AC_3Ath$
بفعالتها	$bC_1C_2AC_3thA$	بفعاليته	$bC_1C_2AC_3yth$
بفعلائنا	$bC_1C_2C_3AAnA$	بفعليتها	$bC_1C_2C_3ythA$
بفعولهما	$bC_1C_2wC_3hmA$	بفعوليات	$bC_1C_2wC_3yAt$
بمفاعلات	$bmC_1AC_2C_3At$	بمفاعلها	$bmC_1AC_2C_3hA$
بمفعولية	$bmC_1C_2wC_3yt$	تتفاعلان	$ttC_1AC_2C_3An$
تتفاعلوا	$ttC_1AC_2C_3wA$	تتفاعلون	$ttC_1AC_2C_3wn$
تتفاعلين	$ttC_1AC_2C_3yn$	تستفعلان	$tstC_1C_2C_3An$
تستفعلوا	$tstC_1C_2C_3wA$	تستفعلون	$tstC_1C_2C_3wn$
تستفعلين	$tstC_1C_2C_3yn$	تفاعلتما	$tC_1AC_2C_3tmA$
تفاعيلهم	$tC_1AC_2yC_3hm$	تفعلانها	$tC_1C_2C_3AnhA$
تفعلانهم	$tC_1C_2C_3Anhm$	تفعلانهن	$tC_1C_2C_3Anhn$
تفعلنهما	$tC_1C_2C_3nhmA$	تفعلونها	$tC_1C_2C_3wnhA$
تفعلونهم	$tC_1C_2C_3wnhm$	تفعلونهن	$tC_1C_2C_3wnhn$
تفعيلهما	$tC_1C_2yC_3hmA$	فافتعلنا	$fAC_1tC_2C_3nA$
فالتفاعل	$fAltC_1AC_2C_3$	فعالاتها	$C_1C_2AC_3AthA$
فعالاتهم	$C_1C_2AC_3Athm$	فعلتاكما	$C_1C_2C_3tAkmA$
فعلتاهما	$C_1C_2C_3tAhmA$	فعلتماها	$C_1C_2C_3tmAhA$
فعلتموها	$C_1C_2C_3tmwhA$	فعلتموهن	$C_1C_2C_3tmwhn$
فعلناكما	$C_1C_2C_3nAkmA$	فعلناهما	$C_1C_2C_3nAhmA$
فعلينيات	$C_1C_2C_3ynyAt$	فعيلاتهم	$C_1C_2yC_3Athm$
كالمفعلة	$kAlmC_1C_2C_3t$	كفعاليته	$kC_1C_2AC_3yth$
لاستفعال	$lAstC_1C_2AC_3$	للافتعال	$llAC_1tC_2AC_3$
للتفعلات	$lltC_1C_2C_3At$	للفاعلين	$llC_1AC_2C_3yn$
للفعالات	$llC_1C_2AC_3At$	للفعليات	$llC_1C_2C_3yAt$
لمفعولتك	$lmC_1C_2wC_3tk$	لمنفعلون	$lmnC_1C_2C_3wn$
مفاعيلهم	$mC_1AC_2yC_3hm$	مفعلاتهم	$mC_1C_2C_3Athm$
مفعلاتهن	$mC_1C_2C_3Athn$	مفعوليات	$mC_1C_2wC_3yAt$
وافعلتهم	$wAC_1C_2C_3thm$	والافعلة	$wAlAC_1C_2C_3t$
والتفعيل	$wAltC_1C_2yC_3$	والفاعلة	$wAlC_1AC_2C_3t$
والفعائل	$wAlC_1C_2AAC_3$	والفعالة	$wAlC_1C_2AC_3t$
والفعلاء	$wAlC_1C_2C_3A'$	والفعيلة	$wAlC_1C_2yC_3t$
والفيعلة	$wAlC_1yC_2C_3t$	والمفاعل	$wAlmC_1AC_2C_3$
وتفعيلات	$wtC_1C_2yC_3At$	وفعاليته	$wC_1C_2AC_3yth$
وفعاليها	$wC_1C_2AC_3yhA$	ومستفعلا	$wmstC_1C_2C_3A$
ومفاعليه	$wmC_1AC_2C_3yh$	ومفعولات	$wmC_1C_2wC_3At$
ويفاعلون	$wyC_1AC_2C_3wn$	يتفاعلان	$ytC_1AC_2C_3An$
يتفاعلوا	$ytC_1AC_2C_3wA$	يتفاعلون	$ytC_1AC_2C_3wn$

table continued on next page

Table A.5 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
يستفعلان	$ystC_1C_2C_3An$	يستفعلني	$ystC_1C_2C_3ny$
يستفعلوا	$ystC_1C_2C_3wA$	يستفعلون	$ystC_1C_2C_3wn$
يفعلانها	$yC_1C_2C_3AnhA$	يفعلانهم	$yC_1C_2C_3Anhm$
يفعلانهن	$yC_1C_2C_3Anhn$	يفعلنهما	$yC_1C_2C_3nhmA$
يفعلوننا	$yC_1C_2C_3wnnA$	يفعلونها	$yC_1C_2C_3wnhA$
يفعلونهم	$yC_1C_2C_3wnhm$	يفعلونهن	$yC_1C_2C_3wnhn$

Table A.6: A List of Compiled Arabic Patterns for $patt_9$

Pattern	Pattern Form	Pattern	Pattern Form
إستفعلتها	$\underline{A}stC_1C_2C_3thA$	إستفعلهما	$\underline{A}stC_1C_2C_3hmA$
افتعالاتك	$AC_1tC_2AC_3Atk$	إفعالاتنا	$\underline{A}C_1C_2AC_3AtnA$
إفعالاتها	$\underline{A}C_1C_2AC_3AthA$	إفعالاتهم	$\underline{A}C_1C_2AC_3Athm$
إفعالاتهن	$\underline{A}C_1C_2AC_3Athn$	إفعالياتك	$\underline{A}C_1C_2AC_3yAtk$
إفعالياته	$\underline{A}C_1C_2AC_3yAth$	إفعاليتان	$\underline{A}C_1C_2AC_3ytAn$
إفعاليتين	$\underline{A}C_1C_2AC_3ytyn$	أفعلائهما	$AC_1C_2C_3AAhmA$
أفعولاتها	$AC_1C_2wC_3AthA$	استفعالات	$AstC_1C_2AC_3At$
استفعالنا	$AstC_1C_2AC_3nA$	استفعالها	$AstC_1C_2AC_3hA$
استفعالهم	$AstC_1C_2AC_3hm$	استفعاليا	$AstC_1C_2AC_3yA$
استفعالية	$AstC_1C_2AC_3yt$	استفعلتكم	$AstC_1C_2C_3tkm$
استفعلتما	$AstC_1C_2C_3tmA$	استفعلتنا	$AstC_1C_2C_3tnA$
استفعلتني	$AstC_1C_2C_3tny$	استفعلتها	$AstC_1C_2C_3thA$
استفعلتهم	$AstC_1C_2C_3thm$	استفعلناه	$AstC_1C_2C_3nAh$
استفعلهما	$AstC_1C_2C_3hmA$	استفعلوكم	$AstC_1C_2C_3wkm$
استفعلوها	$AstC_1C_2C_3whA$	اسفعالاته	$AsC_1C_2AC_3Ath$
افتعالاتك	$AC_1tC_2AC_3Atk$	افتعالاته	$AC_1tC_2AC_3Ath$
افتعالاتي	$AC_1tC_2AC_3Aty$	افتعالهما	$AC_1tC_2AC_3hmA$
افتعاليات	$AC_1tC_2AC_3yAt$	افتعاليها	$AC_1tC_2AC_3yhA$
افتعاليون	$AC_1tC_2AC_3ywn$	افتعاليين	$AC_1tC_2AC_3yyn$
افتعلتهما	$AC_1tC_2C_3thmA$	افتعلناها	$AC_1tC_2C_3nAhA$
افعالاتنا	$AC_1C_2AC_3AtnA$	افعالاتها	$AC_1C_2AC_3AthA$
افعالاتهم	$AC_1C_2AC_3Athm$	افعالياته	$AC_1C_2AC_3yAth$
افعاليتها	$AC_1C_2AC_3ythA$	افعاليتين	$AC_1C_2AC_3ytyn$
افعلائهما	$AC_1C_2C_3AAhmA$	افعلينيات	$AC_1C_2C_3ynyAt$
افعوليتين	$AC_1C_2wC_3ytyn$	الأستفعال	$AlAstC_1C_2AC_3$
الإستفعال	$Al\underline{A}stC_1C_2AC_3$	الإستفعلة	$Al\underline{A}stC_1C_2C_3t$
الإفعالات	$Al\underline{A}C_1C_2AC_3At$	الإفعالية	$Al\underline{A}C_1C_2AC_3yt$
الأفعوالة	$AlAC_1C_2wAC_3t$	الأفعولات	$AlAC_1C_2wC_3At$
الأفعولين	$AlAC_1C_2wC_3yn$	الاستفعال	$AlAstC_1C_2AC_3$
الاستفعلة	$AlAstC_1C_2C_3t$	الاستفعلي	$AlAstC_1C_2C_3y$
الافتعالة	$AlAC_1tC_2AC_3t$	الافتعالى	$AlAC_1tC_2AC_3A$
الافتعالي	$AlAC_1tC_2AC_3y$	الافعالات	$AlAC_1C_2AC_3At$
الافعالان	$AlAC_1C_2AC_3An$	الافعالية	$AlAC_1C_2AC_3yt$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
الافعالين	$AlAC_1C_2AC_3yn$	الافعلاتي	$AlAC_1C_2C_3Aty$
الافعليون	$AlAC_1C_2C_3ywn$	الافعليين	$AlAC_1C_2C_3yyn$
الافعولية	$AlAC_1C_2wC_3yt$	الافعيلية	$AlAC_1C_2yC_3yt$
الانفعالة	$AlAnC_1C_2AC_3t$	الانفعالي	$AlAnC_1C_2AC_3y$
التفاعلات	$AltC_1AC_2C_3At$	التفاعلية	$AltC_1AC_2C_3yt$
التفعالات	$AltC_1C_2AC_3At$	التفعيلات	$AltC_1C_2yC_3At$
التفعيلية	$AltC_1C_2yC_3yt$	التفعيلين	$AltC_1C_2yC_3yn$
الفاعلتان	$AlC_1AC_2C_3tAn$	الفاعلتين	$AlC_1AC_2C_3tyn$
الفاعليات	$AlC_1AC_2C_3yAt$	الفاعليون	$AlC_1AC_2C_3ywn$
الفاعليين	$AlC_1AC_2C_3yyn$	الفاعولية	$AlC_1AC_2wC_3yt$
الفعائلية	$AlC_1C_2AAC_3yt$	الفعالاتي	$AlC_1C_2AC_3Aty$
الفعالتان	$AlC_1C_2AC_3tAn$	الفعالتين	$AlC_1C_2AC_3tyn$
الفعاليات	$AlC_1C_2AC_3yAt$	الفعاليون	$AlC_1C_2AC_3ywn$
الفعاليين	$AlC_1C_2AC_3yyn$	الفعلائية	$AlC_1C_2C_3AAyt$
الفعلانات	$AlC_1C_2C_3AnAt$	الفعلانية	$AlC_1C_2C_3Anyt$
الفعلاوية	$AlC_1C_2C_3Awyt$	الفعلنيون	$AlC_1C_2C_3nywn$
الفعلويون	$AlC_1C_2C_3wywn$	الفعليتان	$AlC_1C_2C_3ytAn$
الفعليتين	$AlC_1C_2C_3ytyn$	الفعلينات	$AlC_1C_2C_3ynAt$
الفعلينية	$AlC_1C_2C_3ynyt$	الفعولتان	$AlC_1C_2wC_3tAn$
الفعولتين	$AlC_1C_2wC_3tyn$	الفعوليات	$AlC_1C_2wC_3yAt$
الفعوليون	$AlC_1C_2wC_3ywn$	الفعوليين	$AlC_1C_2wC_3yyn$
الفعويلية	$AlC_1C_2wyC_3yt$	الفعيلاني	$AlC_1C_2yC_3Any$
الفعيلتان	$AlC_1C_2yC_3tAn$	الفعيلتين	$AlC_1C_2yC_3tyn$
الفعيليات	$AlC_1C_2yC_3yAt$	الفعيليون	$AlC_1C_2yC_3ywn$
الفعيليين	$AlC_1C_2yC_3yyn$	الفمعولية	$AlC_1mC_2wC_3yt$
اللاتفاعل	$AllAtC_1AC_2C_3$	اللافاعلة	$AllAC_1AC_2C_3t$
اللافاعلي	$AllAC_1AC_2C_3y$	اللافعالة	$AllAC_1C_2AC_3t$
اللافعالي	$AllAC_1C_2AC_3y$	اللامفاعل	$AllAmC_1AC_2C_3$
اللامفعلي	$AllAmC_1C_2C_3y$	اللامفعول	$AllAmC_1C_2wC_3$
المتفاعلة	$AlmtC_1AC_2C_3t$	المتفعلات	$AlmtC_1C_2C_3At$
المتفعلان	$AlmtC_1C_2C_3An$	المتفعلون	$AlmtC_1C_2C_3wn$
المتفعلية	$AlmtC_1C_2C_3yt$	المتفعلين	$AlmtC_1C_2C_3yn$
المتفعلين	$AlmtC_1C_2C_3yn$	المتفعلين	$AlmtC_1C_2C_3yn$
المتفعلين	$AlmtC_1C_2C_3yn$	المتمفعلة	$AlmtmC_1C_2C_3t$
المستفعلة	$AlmstC_1C_2C_3t$	المستفعلة	$AlmstC_1C_2C_3t$
المستفعلي	$AlmstC_1C_2C_3y$	المستيفعل	$AlmstyC_1C_2C_3$
المفاعلات	$AlmC_1AC_2C_3At$	المفاعلان	$AlmC_1AC_2C_3An$
المفاعلون	$AlmC_1AC_2C_3wn$	المفاعلية	$AlmC_1AC_2C_3yt$
المفاعلين	$AlmC_1AC_2C_3yn$	المفتاعلة	$AlmC_1tAC_2C_3t$
المفتعلات	$AlmC_1tC_2C_3At$	المفتعلان	$AlmC_1tC_2C_3An$
المفتعلون	$AlmC_1tC_2C_3wn$	المفتعلية	$AlmC_1tC_2C_3yt$
المفتعلين	$AlmC_1tC_2C_3yn$	المفعالان	$AlmC_1C_2AC_3An$
المفعالية	$AlmC_1C_2AC_3yt$	المفعلاتي	$AlmC_1C_2C_3Aty$
المفعلاني	$AlmC_1C_2C_3Any$	المفعلتان	$AlmC_1C_2C_3tAn$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
المفعلتين	$AlmC_1C_2C_3tyn$	المفعليات	$AlmC_1C_2C_3yAt$
المفعليون	$AlmC_1C_2C_3ywn$	المفعليين	$AlmC_1C_2C_3yyn$
المفعولات	$AlmC_1C_2wC_3At$	المفعولات	$AlmC_1C_2wC_3At$
المفعولون	$AlmC_1C_2wC_3wn$	المفعولية	$AlmC_1C_2wC_3yt$
المفعولين	$AlmC_1C_2wC_3yn$	المفيعلية	$AlmC_1yC_2C_3yt$
المنفعلات	$AlmnC_1C_2C_3At$	المنفعلون	$AlmnC_1C_2C_3wn$
انفعالاته	$AnC_1C_2AC_3Ath$	انفعالاتي	$AnC_1C_2AC_3Aty$
انفعالتها	$AnC_1C_2AC_3thA$	بأفعاليات	$bAC_1C_2AC_3yAt$
بإفعاليات	$bAC_1C_2AC_3yAt$	بأفعلياته	$bAC_1C_2C_3yAth$
باستفعالك	$bAstC_1C_2AC_3k$	باستفعاله	$bAstC_1C_2AC_3h$
باستفعلته	$bAstC_1C_2C_3th$	باستفعلتي	$bAstC_1C_2C_3ty$
بافاعيلها	$bAC_1AC_2yC_3hA$	بافتعالات	$bAC_1tC_2AC_3At$
بافتعالنا	$bAC_1tC_2AC_3nA$	بافتعالها	$bAC_1tC_2AC_3hA$
بافتعالهم	$bAC_1tC_2AC_3hm$	بافتعالية	$bAC_1tC_2AC_3yt$
بافعالهما	$bAC_1C_2AC_3hmA$	بافعاليات	$bAC_1C_2AC_3yAt$
بافعلاتها	$bAC_1C_2C_3AthA$	بالأفاعيل	$bAlAC_1AC_2yC_3$
بالإفتعال	$bAlAC_1tC_2AC_3$	بالإفعالة	$bAlAC_1C_2AC_3t$
بالأفعلاء	$bAlAC_1C_2C_3A'$	بالأفعولة	$bAlAC_1C_2wC_3t$
بالافاعيل	$bAlAC_1AC_2yC_3$	بالافتعال	$bAlAC_1tC_2AC_3$
بالافعلية	$bAlAC_1C_2C_3yt$	بالافعولة	$bAlAC_1C_2wC_3t$
بالافعيال	$bAlAC_1C_2yAC_3$	بالانفعال	$bAlAnC_1C_2AC_3$
بالتفاعيل	$bAltC_1AC_2yC_3$	بالتفعلات	$bAltC_1C_2C_3At$
بالتفعلية	$bAltC_1C_2C_3yt$	بالفاعلات	$bAlC_1AC_2C_3At$
بالفاعلان	$bAlC_1AC_2C_3An$	بالفاعلية	$bAlC_1AC_2C_3yt$
بالفاعلين	$bAlC_1AC_2C_3yn$	بالفاعيلة	$bAlC_1AC_2yC_3t$
بالفعالات	$bAlC_1C_2AC_3At$	بالفعالات	$bAlC_1C_2AC_3At$
بالفعالية	$bAlC_1C_2AC_3yt$	بالفعلوات	$bAlC_1C_2C_3wAt$
بالفعليات	$bAlC_1C_2C_3yAt$	بالفعليين	$bAlC_1C_2C_3yyn$
بالفعولات	$bAlC_1C_2wC_3At$	بالفعولية	$bAlC_1C_2wC_3yt$
بالفعيلات	$bAlC_1C_2yC_3At$	بالفعيلات	$bAlC_1C_2yC_3At$
بالفعيلين	$bAlC_1C_2yC_3yn$	بالفواعيل	$bAlC_1wAC_2yC_3$
بالفيعولة	$bAlC_1yC_2wC_3t$	بالمستفعل	$bAlmstC_1C_2C_3$
بالمفاعلة	$bAlmC_1AC_2C_3t$	بالمفاعيل	$bAlmC_1AC_2yC_3$
بالمفتعلن	$bAlmC_1tC_2C_3n$	بالمفعلات	$bAlmC_1C_2C_3At$
بالمفعلان	$bAlmC_1C_2C_3An$	بالمفعلية	$bAlmC_1C_2C_3yt$
بالمفعلين	$bAlmC_1C_2C_3yn$	بانفعالات	$bAnC_1C_2AC_3At$
بانفعالها	$bAnC_1C_2AC_3hA$	بتفاعيلها	$btC_1AC_2yC_3hA$
بتفعليتها	$btC_1C_2C_3ythA$	بتفعيلاته	$btC_1C_2yC_3Ath$
بتفعيلهما	$btC_1C_2yC_3hmA$	بفاعلاتنا	$bC_1AC_2C_3AtnA$
بفاعلاتها	$bC_1AC_2C_3AthA$	بفاعليتها	$bC_1AC_2C_3ythA$
بفعالاتنا	$bC_1C_2AC_3AtnA$	بفعالاتها	$bC_1C_2AC_3AthA$
بفعالاتهم	$bC_1C_2AC_3Athm$	بفعالتهما	$bC_1C_2AC_3thmA$
بفعالياته	$bC_1C_2AC_3yAth$	بفعاليتها	$bC_1C_2AC_3ythA$
بفعلياتها	$bC_1C_2C_3yAthA$	بفعلياتهم	$bC_1C_2C_3yAthm$
بفعولاتها	$bC_1C_2wC_3AthA$	بفعوليتها	$bC_1C_2wC_3ythA$
بفعيلاتها	$bC_1C_2yC_3AthA$	بمستفعلات	$bmstC_1C_2C_3At$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
بمستفعلهم	$bmstC_1C_2C_3hm$	بمفاعلاته	$bmC_1AC_2C_3Ath$
بمفاعلاتي	$bmC_1AC_2C_3Aty$	بمفاعلتنا	$bmC_1AC_2C_3tnA$
بمفاعلتها	$bmC_1AC_2C_3thA$	بمفاعلتهم	$bmC_1AC_2C_3thm$
بمفاعلتهن	$bmC_1AC_2C_3thn$	بمفاعيلها	$bmC_1AC_2yC_3hA$
بمفتعلاته	$bmC_1tC_2C_3Ath$	بمفعلاتها	$bmC_1C_2C_3AthA$
بمفعليتها	$bmC_1C_2C_3ythA$	بمفعولتين	$bmC_1C_2wC_3tym$
بمفعوليات	$bmC_1C_2wC_3yAt$	بمفعوليته	$bmC_1C_2wC_3yth$
تستفعلونه	$tstC_1C_2C_3wnh$	تفاعلاتها	$tC_1AC_2C_3AthA$
تفاعلاتهم	$tC_1AC_2C_3Athm$	تفاعلينني	$tC_1AC_2C_3ymny$
تفتعلونها	$tC_1tC_2C_3wnhA$	تفعلانهما	$tC_1C_2C_3AnhmA$
تفعلونهما	$tC_1C_2C_3wnhmA$	تفعيلاتكم	$tC_1C_2yC_3Atkm$
تفعيلاتنا	$tC_1C_2yC_3AtnA$	تفعيلاتها	$tC_1C_2yC_3AthA$
تفعيلاتهم	$tC_1C_2yC_3Athm$	تفعيلياته	$tC_1C_2yC_3yAth$
تفعيليتها	$tC_1C_2yC_3ythA$	سافتعلهما	$sAC_1tC_2C_3hmA$
ستستفعلين	$ststC_1C_2C_3yn$	سنتفاعلها	$sntC_1AC_2C_3hA$
سيتفاعلان	$sytC_1AC_2C_3An$	سيتفاعلون	$sytC_1AC_2C_3wn$
سيستفعلها	$systC_1C_2C_3hA$	سيستفعلون	$systC_1C_2C_3wn$
سيفعلوننا	$syC_1C_2C_3wnnA$	سيفعلونهم	$syC_1C_2C_3wnhm$
فاستفعاله	$fAstC_1C_2AC_3h$	فافتعالهم	$fAC_1tC_2AC_3hm$
فافتعلتها	$fAC_1tC_2C_3thA$	فافعالاته	$fAC_1C_2AC_3Ath$
فالأفعلاء	$fAlAC_1C_2C_3A'$	فالأفعولة	$fAlAC_1C_2wC_3t$
فالافتعال	$fAlAC_1tC_2AC_3$	فالافعلاء	$fAlAC_1C_2C_3A'$
فالانفعال	$fAlAnC_1C_2AC_3$	فالتفاعيل	$fAltC_1AC_2yC_3$
فالفاعلات	$fAlC_1AC_2C_3At$	فالفاعلون	$fAlC_1AC_2C_3wn$
فالفاعلين	$fAlC_1AC_2C_3yn$	فالفعالات	$fAlC_1C_2AC_3At$
فالفعالية	$fAlC_1C_2AC_3yt$	فالفعليات	$fAlC_1C_2C_3yAt$
فالفعيلان	$fAlC_1C_2yC_3An$	فالفعيلون	$fAlC_1C_2yC_3wn$
فالفواعيل	$fAlC_1wAC_2yC_3$	فالمفاعلة	$fAlmC_1AC_2C_3t$
فالمفاعيل	$fAlmC_1AC_2yC_3$	فالمفعلات	$fAlmC_1C_2C_3At$
فالمفعلان	$fAlmC_1C_2C_3An$	فالمفعلون	$fAlmC_1C_2C_3wn$
فالمفعلية	$fAlmC_1C_2C_3yt$	فالمفعولة	$fAlmC_1C_2wC_3t$
فبالإفعال	$fbAlAC_1C_2AC_3$	فبالمفاعل	$fbAlmC_1AC_2C_3$
فتستفعلنا	$ftstC_1C_2C_3nA$	فتستفعلوا	$ftstC_1C_2C_3wA$
فعائليتنا	$C_1C_2AAC_3ytnA$	فعالاتهما	$C_1C_2AC_3AthmA$
فعالياتها	$C_1C_2AC_3yAthA$	فعلانيتها	$C_1C_2C_3AnythA$
فعلانيتها	$C_1C_2C_3AnythA$	فعلتموهما	$C_1C_2C_3tmwhmA$
فعولياتنا	$C_1C_2wC_3yAtnA$	فعولياتها	$C_1C_2wC_3yAthA$
فعولياتهم	$C_1C_2wC_3yAthm$	فعوليتهما	$C_1C_2wC_3ythmA$
ففاعليتهم	$fC_1AC_2C_3ythm$	ففعلتموها	$C_1fC_2C_3tmwhA$
فللمفعلين	$fllmC_1C_2C_3yn$	فمفتعلوها	$fmC_1tC_2C_3whA$
فيتفاعلون	$fytC_1AC_2C_3wn$	فيستفعلها	$fystC_1C_2C_3hA$
فيستفعلون	$fystC_1C_2C_3wn$	فيستفعلون	$fystC_1C_2C_3wn$
فيفتعلوها	$fyC_1tC_2C_3whA$	فيفعلونها	$fyC_1C_2C_3wnhA$
كافتعالنا	$kAC_1tC_2AC_3nA$	كالافتعال	$kAlAC_1tC_2AC_3$
كالانفعال	$kAlAnC_1C_2AC_3$	كالفاعلية	$kAlC_1AC_2C_3yt$
كالفاعلين	$kAlC_1AC_2C_3yn$	كالفعالات	$kAlC_1C_2AC_3At$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
كالفعالية	$kAlC_1C_2AC_3yt$	كالفعالين	$kAlC_1C_2AC_3yn$
كالفعولية	$kAlC_1C_2wC_3yt$	كالفعيلات	$kAlC_1C_2yC_3At$
كالمتفعلة	$kAlmtC_1C_2C_3t$	كالمستفعل	$kAlmstC_1C_2C_3$
كالمفاعيل	$kAlmC_1AC_2yC_3$	كالمفعلات	$kAlmC_1C_2C_3At$
كالمفعلان	$kAlmC_1C_2C_3An$	كالمفعلين	$kAlmC_1C_2C_3yn$
كالمفعولة	$kAlmC_1C_2wC_3t$	كفعلوانين	$kC_1C_2C_3wAnyn$
كمستفعلات	$kmstC_1C_2C_3At$	كمستفعلين	$kmstC_1C_2C_3yn$
لإستفعالا	$lAstC_1C_2AC_3A$	لإفعالاته	$lAC_1C_2AC_3Ath$
لأفعالهما	$lAC_1C_2AC_3hmA$	لإفعاليات	$lAC_1C_2AC_3yAt$
لأفعلاتنا	$lAC_1C_2C_3AtnA$	لاافعيلية	$lAAC_1C_2yC_3yt$
لاستفتعال	$lAstC_1tC_2AC_3$	لاستفعالا	$lAstC_1C_2AC_3A$
لاستفعاله	$lAstC_1C_2AC_3h$	لافتعالات	$lAC_1tC_2AC_3At$
لافتعالنا	$lAC_1tC_2AC_3nA$	لافتعالها	$lAC_1tC_2AC_3hA$
لافتعالهم	$lAC_1tC_2AC_3hm$	لافتعالية	$lAC_1tC_2AC_3yt$
لافتعلناه	$lAC_1tC_2C_3nAh$	لافعالهما	$lAC_1C_2AC_3hmA$
لافعاليات	$lAC_1C_2AC_3yAt$	لامتفاعلة	$lAmtC_1AC_2C_3t$
لانفعالات	$lAnC_1C_2AC_3At$	لتفاعلاته	$ltC_1AC_2C_3Ath$
لتفاعلهما	$ltC_1AC_2C_3hmA$	لتفعلاتنا	$ltC_1C_2C_3AtnA$
لفاعلاتها	$lC_1AC_2C_3AthA$	لفاعليتها	$lC_1AC_2C_3ythA$
لفعالاتنا	$lC_1C_2AC_3AtnA$	لفعالاتها	$lC_1C_2AC_3AthA$
لفعلياتنا	$lC_1C_2C_3yAtnA$	لفعيلاتها	$lC_1C_2yC_3AthA$
لفواعيلها	$lC_1wAC_2yC_3hA$	للإستفعال	$llAstC_1C_2AC_3$
للإفعالات	$llAC_1C_2AC_3At$	للإفعالية	$llAC_1C_2AC_3yt$
للاستفعال	$llAstC_1C_2AC_3$	للافعالات	$llAC_1C_2AC_3At$
للتفاعلات	$lltC_1AC_2C_3At$	للتفعيلات	$lltC_1C_2yC_3At$
للفاعلتين	$llC_1AC_2C_3tyn$	للفعاليات	$llC_1C_2AC_3yAt$
للفعاليين	$llC_1C_2AC_3yyn$	للفعلانات	$llC_1C_2C_3AnAt$
للفعلانية	$llC_1C_2C_3Anyt$	للفعليتين	$llC_1C_2C_3ytyn$
للفعوليات	$llC_1C_2wC_3yAt$	للفعوليين	$llC_1C_2wC_3yyn$
للمتفعلات	$llmtC_1C_2C_3At$	للمتفعلين	$llmtC_1C_2C_3yn$
للمستفعلة	$llmstC_1C_2C_3t$	للمفاعلات	$llmC_1AC_2C_3At$
للمفاعلين	$llmC_1AC_2C_3yn$	للمفتعلات	$llmC_1tC_2C_3At$
للمفتعلين	$llmC_1tC_2C_3yn$	للمفعلاتي	$llmC_1C_2C_3Aty$
للمفعلتين	$llmC_1C_2C_3tyn$	للمفعولات	$llmC_1C_2wC_3At$
للمفعولية	$llmC_1C_2wC_3yt$	للمفعولين	$llmC_1C_2wC_3yn$
لمستفعلات	$lmstC_1C_2C_3At$	لمفاعالات	$lmC_1AC_2AC_3At$
لمفاعلتنا	$lmC_1AC_2C_3tnA$	لمفاعلتها	$lmC_1AC_2C_3thA$
لمفاعلتهم	$lmC_1AC_2C_3thm$	لمفتعلاته	$lmC_1tC_2C_3Ath$
لمفعلاتها	$lmC_1C_2C_3AthA$	لمفعلياتي	$lmC_1C_2C_3yAty$
لمفعولاته	$lmC_1C_2wC_3Ath$	ليتفاعلوا	$lytC_1AC_2C_3wA$
ليستفعلها	$lystC_1C_2C_3hA$	ليستفعلوا	$lystC_1C_2C_3wA$
ليفعلوهما	$lyC_1C_2C_3whmA$	متفاعلتان	$mtC_1AC_2C_3tAn$
متفاعلتين	$mtC_1AC_2C_3tyn$	متفعلاتها	$mtC_1C_2C_3AthA$
متفعلاتهم	$mtC_1C_2C_3Athm$	مستفعلتين	$mstC_1C_2C_3tyn$
مستفعلينا	$mstC_1C_2C_3ynA$	مستفعليهم	$mstC_1C_2C_3yhm$
مفاعلاتنا	$mC_1AC_2C_3AtnA$	مفاعلاتها	$mC_1AC_2C_3AthA$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
مفاعلاتهم	$mC_1AC_2C_3Athm$	مفاعلتهما	$mC_1AC_2C_3thmA$
مفتعلاتنا	$mC_1tC_2C_3AtnA$	مفتعلاتها	$mC_1tC_2C_3AthA$
مفتعلاتهم	$mC_1tC_2C_3Athm$	مفتعلاتهن	$mC_1tC_2C_3Athn$
مفعاليتها	$mC_1C_2AC_3ythA$	مفعلاتهما	$mC_1C_2C_3AthmA$
مفعلياتنا	$mC_1C_2C_3yAtnA$	مفعولاتنا	$mC_1C_2wC_3AtnA$
مفعولاتها	$mC_1C_2wC_3AthA$	مفعولاتهم	$mC_1C_2wC_3Athm$
مفعولاتية	$mC_1C_2wC_3Atyt$	مفعولياته	$mC_1C_2wC_3yAth$
مفعوليتنا	$mC_1C_2wC_3ytnA$	مفعوليتها	$mC_1C_2wC_3ythA$
مفعوليتهم	$mC_1C_2wC_3ythm$	مفوعلاتها	$mC_1wC_2C_3AthA$
منفعلاتهم	$mnC_1C_2C_3Athm$	واستفعلها	$wAstC_1C_2C_3hA$
وافاعيلنا	$wAC_1AC_2yC_3nA$	وافاعيلهم	$wAC_1AC_2yC_3hm$
وإفعالاتك	$wAC_1C_2AC_3Atk$	وإفعالاته	$wAC_1C_2AC_3Ath$
وافعالهما	$wAC_1C_2AC_3hmA$	وافعاليات	$wAC_1C_2AC_3yAt$
وافعاليون	$wAC_1C_2AC_3ywn$	وإفعاليين	$wAC_1C_2AC_3yym$
وافعلاءنا	$wAC_1C_2C_3A'nA$	وافعولاته	$wAC_1C_2wC_3Ath$
واستفعالا	$wAstC_1C_2AC_3A$	واستفعاله	$wAstC_1C_2AC_3h$
واستفعالي	$wAstC_1C_2AC_3y$	واستفعلات	$wAstC_1C_2C_3At$
واستفعلتم	$wAstC_1C_2C_3tm$	واستفعلته	$wAstC_1C_2C_3th$
واستفعلنا	$wAstC_1C_2C_3nA$	واستفعلها	$wAstC_1C_2C_3hA$
واستفعلهم	$wAstC_1C_2C_3hm$	واستفعلوا	$wAstC_1C_2C_3wA$
واستفعلوه	$wAstC_1C_2C_3wh$	وافاعيلها	$wAC_1AC_2yC_3hA$
وافتعالات	$wAC_1tC_2AC_3At$	وافتعالتك	$wAC_1tC_2AC_3tk$
وافتعالته	$wAC_1tC_2AC_3th$	وافتعالنا	$wAC_1tC_2AC_3nA$
وافتعالها	$wAC_1tC_2AC_3hA$	وافتعالهم	$wAC_1tC_2AC_3hm$
وافتعالهن	$wAC_1tC_2AC_3hn$	وافتعاليا	$wAC_1tC_2AC_3yA$
وافتعالية	$wAC_1tC_2AC_3yt$	وافتعلتها	$wAC_1tC_2C_3thA$
وافتعلوها	$wAC_1tC_2C_3whA$	وافعاليات	$wAC_1C_2AC_3yAt$
وافعاليته	$wAC_1C_2AC_3yth$	وافعاليون	$wAC_1C_2AC_3ywn$
وافعلائها	$wAC_1C_2C_3AAhA$	وافعلاتها	$wAC_1C_2C_3AthA$
وافعلاتهم	$wAC_1C_2C_3Athm$	وافعلمائة	$wAC_1C_2C_3mAAt$
والأفاعيل	$wAlAC_1AC_2yC_3$	والأفعالي	$wAlAC_1C_2AC_3y$
والإفعالي	$wAlAC_1C_2AC_3y$	والأفعلاء	$wAlAC_1C_2C_3A'$
والافاعيل	$wAlAC_1AC_2yC_3$	والافتعال	$wAlAC_1tC_2AC_3$
والافتعال	$wAlAC_1tC_2AC_3$	والافعالة	$wAlAC_1C_2AC_3t$
والافعالي	$wAlAC_1C_2AC_3y$	والافعلاء	$wAlAC_1C_2C_3A'$
والافعلات	$wAlAC_1C_2C_3At$	والافعلاك	$wAlAC_1C_2C_3Ak$
والافعلية	$wAlAC_1C_2C_3yt$	والافعيال	$wAlAC_1C_2yAC_3$
والانفعال	$wAlAnC_1C_2AC_3$	والتفاعلي	$wAltC_1AC_2C_3y$
والتفاعيل	$wAltC_1AC_2yC_3$	والتفعلات	$wAltC_1C_2C_3At$
والتفعلية	$wAltC_1C_2C_3yt$	والتفعيلة	$wAltC_1C_2yC_3t$
والتفعيلي	$wAltC_1C_2yC_3y$	والفاعلات	$wAlC_1AC_2C_3At$
والفاعلان	$wAlC_1AC_2C_3An$	والفاعلون	$wAlC_1AC_2C_3wn$
والفاعلية	$wAlC_1AC_2C_3yt$	والفاعلين	$wAlC_1AC_2C_3yn$
والفاعولي	$wAlC_1AC_2wC_3y$	والفعائلة	$wAlC_1C_2AAC_3t$
والفعائلي	$wAlC_1C_2AAC_3y$	والفعالات	$wAlC_1C_2AC_3At$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
والفعالون	$wAlC_1C_2AC_3wn$	والفعالية	$wAlC_1C_2AC_3yt$
والفماليل	$wAlC_1C_2AC_3yl$	والفعالين	$wAlC_1C_2AC_3yn$
والفعاولة	$wAlC_1C_2AwC_3t$	والفعايلة	$wAlC_1C_2AyC_3t$
والفعلاتي	$wAlC_1C_2C_3Aty$	والفعلالي	$wAlC_1C_2C_3Aly$
والفعلانة	$wAlC_1C_2C_3Ant$	والفعلاني	$wAlC_1C_2C_3Any$
والفعلاوي	$wAlC_1C_2C_3Awy$	والفعلتين	$wAlC_1C_2C_3tyn$
والفعلوات	$wAlC_1C_2C_3wAt$	والفعليات	$wAlC_1C_2C_3yAt$
والفعليون	$wAlC_1C_2C_3ywn$	والفعليين	$wAlC_1C_2C_3yyn$
والفعولات	$wAlC_1C_2wC_3At$	والفعولية	$wAlC_1C_2wC_3yt$
والفعولين	$wAlC_1C_2wC_3yn$	والفعيلات	$wAlC_1C_2yC_3At$
والفعيلون	$wAlC_1C_2yC_3wn$	والفعيلية	$wAlC_1C_2yC_3yt$
والفعيلين	$wAlC_1C_2yC_3yn$	والفواعلي	$wAlC_1wAC_2C_3y$
والفواعيل	$wAlC_1wAC_2yC_3$	والفيعلية	$wAlC_1yC_2C_3yt$
والفيعولة	$wAlC_1yC_2wC_3t$	والمتفاعل	$wAlmtC_1AC_2C_3$
والمتفعلة	$wAlmtC_1C_2C_3t$	والمتفعلن	$wAlmtC_1C_2C_3n$
والمتفعيل	$wAlmtC_1C_2yC_3$	والمتمفعل	$wAlmtmC_1C_2C_3$
والمستفعل	$wAlmstC_1C_2C_3$	والمفاعلة	$wAlmC_1AC_2C_3t$
والمفاعيل	$wAlmC_1AC_2yC_3$	والمفتعلة	$wAlmC_1tC_2C_3t$
والمفتعلن	$wAlmC_1tC_2C_3n$	والمفتعلي	$wAlmC_1tC_2C_3y$
والمفعالة	$wAlmC_1C_2AC_3t$	والمفعالي	$wAlmC_1C_2AC_3y$
والمفعلات	$wAlmC_1C_2C_3At$	والمفعلون	$wAlmC_1C_2C_3wn$
والمفعلية	$wAlmC_1C_2C_3yt$	والمفعلين	$wAlmC_1C_2C_3yn$
والمفعولة	$wAlmC_1C_2wC_3t$	والمفعولي	$wAlmC_1C_2wC_3y$
والمفعيلي	$wAlmC_1C_2yC_3y$	والمنفعلة	$wAlmnC_1C_2C_3t$
والموفعال	$wAlmwC_1C_2AC_3$	واليافعيل	$wAlyAC_1C_2yC_3$
وانفعالات	$wAnC_1C_2AC_3At$	وانفعالها	$wAnC_1C_2AC_3hA$
وانفعالية	$wAnC_1C_2AC_3yt$	وبإفعالهم	$wbAC_1C_2AC_3hm$
وبإفعالية	$wbAC_1C_2AC_3yt$	وباستفعال	$wbAstC_1C_2AC_3$
وبافتعاله	$wbAC_1tC_2AC_3h$	وبافعالهم	$wbAC_1C_2AC_3hm$
وبافعالية	$wbAC_1C_2AC_3yt$	وبافعلتها	$wbAC_1C_2C_3thA$
وبالأفعال	$wbAlAC_1C_2AC_3$	وبالافعال	$wbAlAC_1C_2AC_3$
وبالافعول	$wbAlAC_1C_2wC_3$	وبالتفاعل	$wbAltC_1AC_2C_3$
وبالتفعيل	$wbAltC_1C_2yC_3$	وبالفعائل	$wbAlC_1C_2AAC_3$
وبالفعالة	$wbAlC_1C_2AC_3t$	وبالفعلات	$wbAlC_1C_2C_3At$
وبالفعلان	$wbAlC_1C_2C_3An$	وبالفعلية	$wbAlC_1C_2C_3yt$
وبالفعيلة	$wbAlC_1C_2yC_3t$	وبالمفاعل	$wbAlmC_1AC_2C_3$
وبالمفتعل	$wbAlmC_1tC_2C_3$	وبالمفعلة	$wbAlmC_1C_2C_3t$
وبتفعالنا	$wbtC_1C_2AC_3nA$	وبتفعيلته	$wbtC_1C_2yC_3th$
وبتفعيلها	$wbtC_1C_2yC_3hA$	وبفاعلتها	$wbC_1AC_2C_3thA$
وبفعالتها	$wbC_1C_2AC_3thA$	وبفعالتهم	$wbC_1C_2AC_3thm$
وبفعلاتهم	$wbC_1C_2C_3Athm$	وبفعليتها	$wbC_1C_2C_3ythA$
وبفعيلتها	$wbC_1C_2yC_3thA$	وبمفاعلات	$wbmC_1AC_2C_3At$
وبمفعولية	$wbmC_1C_2wC_3yt$	وتفاعيلنا	$wtC_1AC_2yC_3nA$
وتفاعيلها	$wtC_1AC_2yC_3hA$	وتفاعيلهم	$wtC_1AC_2yC_3hm$
وتفعلاتهم	$wtC_1C_2C_3Athm$	وتفعيلاتك	$wtC_1C_2yC_3Atk$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
وتفعيلاته	$wtC_1C_2yC_3Ath$	وتفعيلهما	$wtC_1C_2yC_3hmA$
وفاعلاتها	$wC_1AC_2C_3AthA$	وفاعلاتهم	$wC_1AC_2C_3Athm$
وفاعلتهما	$wC_1AC_2C_3thmA$	وفاعليتها	$wC_1AC_2C_3ythA$
وفعالاتنا	$wC_1C_2AC_3AtnA$	وفعالاتها	$wC_1C_2AC_3AthA$
وفعالاتهم	$wC_1C_2AC_3Athm$	وفعالاتية	$wC_1C_2AC_3Atyt$
وفعالتهما	$wC_1C_2AC_3thmA$	وفعالمائة	$wC_1C_2AC_3mAAt$
وفعالياته	$wC_1C_2AC_3yAth$	وفعاليتها	$wC_1C_2AC_3ythA$
وفعاليتهم	$wC_1C_2AC_3ythm$	وفعالييها	$wC_1C_2AC_3yyhA$
وفعلانيات	$wC_1C_2C_3AnyAt$	وفعلانيته	$wC_1C_2C_3Anyth$
وفعلانيين	$wC_1C_2C_3Anyyn$	وفعلتاهما	$wC_1C_2C_3tAhmA$
وفعلتموها	$wC_1C_2C_3tmwhA$	وفعلياتها	$wC_1C_2C_3yAthA$
وفعلياتهم	$wC_1C_2C_3yAthm$	وفعولاتها	$wC_1C_2wC_3AthA$
وفعولاتهم	$wC_1C_2wC_3Athm$	وفعوليتها	$wC_1C_2wC_3ythA$
وفعوليتهم	$wC_1C_2wC_3ythm$	وفعيلاتها	$wC_1C_2yC_3AthA$
وفعيليتها	$wC_1C_2yC_3ythA$	وفواعيلنا	$wC_1wAC_2yC_3nA$
وفواعيلها	$wC_1wAC_2yC_3hA$	وفواعيلهم	$wC_1wAC_2yC_3hm$
وفيعولتها	$wC_1yC_2wC_3thA$	وكإفعالات	$wkAC_1C_2AC_3At$
وكافعالات	$wkAC_1C_2AC_3At$	وكالفعلات	$wkAlC_1C_2C_3At$
ولإفعالنا	$wlAC_1C_2AC_3nA$	ولأفعلنهم	$wlAC_1C_2C_3nhm$
ولتفاعلهم	$wltC_1AC_2C_3hm$	ولتفعيلها	$wltC_1C_2yC_3hA$
ولفعاللها	$wlC_1C_2AAC_3hA$	ولفعالتها	$wlC_1C_2AC_3thA$
ولفعلاتها	$wlC_1C_2C_3AthA$	وللافتعال	$wllAC_1tC_2AC_3$
وللتفعلات	$wlltC_1C_2C_3At$	وللفاعلين	$wllC_1AC_2C_3ym$
وللفعالات	$wllC_1C_2AC_3At$	وللمفاعلة	$wllmC_1AC_2C_3t$
وللمفعلات	$wllmC_1C_2C_3At$	وللمفعلين	$wllmC_1C_2C_3yn$
ولمفاعليه	$wlmC_1AC_2C_3yh$	ولمفعلتها	$wlmC_1C_2C_3thA$
ولنفعلنهم	$wlnC_1C_2C_3nhm$	ومتفعلاته	$wmtC_1C_2C_3Ath$
ومتفعليين	$wmtC_1C_2C_3yyn$	ومستفعلات	$wmstC_1C_2C_3At$
ومستفعلهم	$wmstC_1C_2C_3hm$	ومستفعلين	$wmstC_1C_2C_3yn$
ومفاعلاته	$wmC_1AC_2C_3Ath$	ومفاعلاتي	$wmC_1AC_2C_3Aty$
ومفاعلتها	$wmC_1AC_2C_3thA$	ومفاعلتهم	$wmC_1AC_2C_3thm$
ومفاعليها	$wmC_1AC_2C_3yhA$	ومفاعيلهم	$wmC_1AC_2yC_3hm$
ومفتعلاته	$wmC_1tC_2C_3Ath$	ومفتعلوها	$wmC_1tC_2C_3whA$
ومفعلاتنا	$wmC_1C_2C_3AtnA$	ومفعلاتها	$wmC_1C_2C_3AthA$
ومفعلاتهم	$wmC_1C_2C_3Athm$	ومفعلانات	$wmC_1C_2C_3AnAt$
ومفعولاته	$wmC_1C_2wC_3Ath$	ومنفعلاته	$wmnC_1C_2C_3Ath$
ويتفاعلان	$wytC_1AC_2C_3An$	ويتفاعلون	$wytC_1AC_2C_3wn$
ويتفعلونه	$wytC_1C_2C_3wnh$	ويستفعلان	$wystC_1C_2C_3An$
ويستفعلني	$wystC_1C_2C_3ny$	ويستفعلها	$wystC_1C_2C_3hA$
ويستفعلوا	$wystC_1C_2C_3wA$	ويستفعلون	$wystC_1C_2C_3wn$
ويفاعلوهم	$wyC_1AC_2C_3whm$	ويفتعلونه	$wyC_1tC_2C_3wnh$
ويفعلانها	$wyC_1C_2C_3AnhA$	ويفعلونها	$wyC_1C_2C_3wnhA$
ويفعلونهم	$wyC_1C_2C_3wnhm$	يتفاعلهما	$ytC_1AC_2C_3hmA$
يتفاعلونك	$ytC_1AC_2C_3wnk$	يتفاعلونه	$ytC_1AC_2C_3wnh$
يتفعلونها	$ytC_1C_2C_3wnhA$	يستفعلنها	$ystC_1C_2C_3nhA$
يستفعلونه	$ystC_1C_2C_3wnh$	يفاعلونكم	$yC_1AC_2C_3wnkm$

table continued on next page

Table A.6 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
يفاعلوننا	$yC_1AC_2C_3wnnA$	يفاعلونني	$yC_1AC_2C_3wnny$
يفاعلونها	$yC_1AC_2C_3wnhA$	يفاعلونهم	$yC_1AC_2C_3wnhm$
يفتعلونها	$yC_1tC_2C_3wnhA$	يفتعلونهم	$yC_1tC_2C_3wnhm$
يفعلانهما	$yC_1C_2C_3AnhmA$	يفعلونهما	$yC_1C_2C_3wnhmA$

Table A.7: A List of Compiled Arabic Patterns for $patt_{10}$

Pattern	Pattern Form	Pattern	Pattern Form
استفعالاته	$AstC_1C_2AC_3Ath$	استفعالهما	$AstC_1C_2AC_3hmA$
استفعلاتهم	$AstC_1C_2C_3Athm$	استفعلناها	$AstC_1C_2C_3nAhA$
افتعالاتكم	$AC_1tC_2AC_3Atkm$	افتعالاتنا	$AC_1tC_2AC_3AtnA$
افتعالاتها	$AC_1tC_2AC_3AthA$	افتعالاتهم	$AC_1tC_2AC_3Athm$
افتعالياته	$AC_1tC_2AC_3yAth$	افتعاليتها	$AC_1tC_2AC_3ythA$
افعالياتها	$AC_1C_2AC_3yAthA$	افعالياتهم	$AC_1C_2AC_3yAthm$
الاستفعالي	$AlAstC_1C_2AC_3y$	الاستفعلات	$AlAstC_1C_2C_3At$
الاستفعلية	$AlAstC_1C_2C_3yt$	الافتعالات	$AlAC_1tC_2AC_3At$
الافتعالية	$AlAC_1tC_2AC_3yt$	الافتعالين	$AlAC_1tC_2AC_3yn$
الافعاليات	$AlAC_1C_2AC_3yAt$	الافعاليون	$AlAC_1C_2AC_3ywn$
الافعاليين	$AlAC_1C_2AC_3yyn$	الافعيالية	$AlAC_1C_2yAC_3yt$
الانفعالات	$AlAnC_1C_2AC_3At$	الانفعالية	$AlAnC_1C_2AC_3yt$
التفاعليات	$AltC_1AC_2C_3yAt$	التفاعيلية	$AltC_1AC_2yC_3yt$
التفعيليات	$AltC_1C_2yC_3yAt$	التفعيليان	$AltC_1C_2yC_3yAn$
التفعيليون	$AltC_1C_2yC_3ywn$	التفعيليين	$AltC_1C_2yC_3yyn$
الفاعليتين	$AlC_1AC_2C_3ytyn$	الفعاليتان	$AlC_1C_2AC_3ynAt$
الفعلانيات	$AlC_1C_2C_3AnyAt$	الفعلانيون	$AlC_1C_2C_3Anywn$
الفعلانيين	$AlC_1C_2C_3Anyyn$	الفعلوانية	$AlC_1C_2C_3wAnyt$
الفعلينيات	$AlC_1C_2C_3ynyAt$	الفعلينيون	$AlC_1C_2C_3ynywn$
الفعليني	$AlC_1C_2C_3ynyyn$	الفعيلانية	$AlC_1C_2yC_3Anyt$
الفعيليتين	$AlC_1C_2yC_3ytyn$	الفواعليين	$AlC_1wAC_2C_3yyn$
الفواعيلية	$AlC_1wAC_2yC_3yt$	المتفاعلات	$AlmtC_1AC_2C_3At$
المتفاعلان	$AlmtC_1AC_2C_3An$	المتفاعلون	$AlmtC_1AC_2C_3wn$
المتفاعلين	$AlmtC_1AC_2C_3yn$	المستفعلات	$AlmstC_1C_2C_3At$
المستفعلان	$AlmstC_1C_2C_3An$	المستفعلون	$AlmstC_1C_2C_3wn$
المستفعلية	$AlmstC_1C_2C_3yt$	المستفعلين	$AlmstC_1C_2C_3yn$
المفاعلتين	$AlmC_1AC_2C_3tyn$	المفاعيلية	$AlmC_1AC_2yC_3yt$
المفتاعلات	$AlmC_1tAC_2C_3At$	المفتعلتان	$AlmC_1tC_2C_3tAn$
المفتعلتين	$AlmC_1tC_2C_3tyn$	المفعالتين	$AlmC_1C_2AC_3tyn$
المفعاليات	$AlmC_1C_2AC_3yAt$	المفعاليون	$AlmC_1C_2AC_3ywn$
المفعاليين	$AlmC_1C_2AC_3yyn$	المفعلاتين	$AlmC_1C_2C_3Atyn$
المفعلانات	$AlmC_1C_2C_3AnAt$	المفعلانية	$AlmC_1C_2C_3Anyt$
المفعولاتي	$AlmC_1C_2wC_3Aty$	المفعولتان	$AlmC_1C_2wC_3tAn$
المفعولتين	$AlmC_1C_2wC_3tyn$	المفعوليات	$AlmC_1C_2wC_3yAt$
المفعوليين	$AlmC_1C_2wC_3yyn$	المنفعلتان	$AlmnC_1C_2C_3tAn$

table continued on next page

Table A.7 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
انفعالاتها	$AnC_1C_2AC_3AthA$	انفعالاتهم	$AnC_1C_2AC_3Athm$
باستفعالات	$bAstC_1C_2AC_3At$	باستفعالنا	$bAstC_1C_2AC_3nA$
باستفعالها	$bAstC_1C_2AC_3hA$	باستفعالية	$bAstC_1C_2AC_3yt$
باستفعلتها	$bAstC_1C_2C_3thA$	باستفعلتهم	$bAstC_1C_2C_3thm$
بافتعالاته	$bAC_1tC_2AC_3Ath$	بافتعالهما	$bAC_1tC_2AC_3hmA$
بافتعاليات	$bAC_1tC_2AC_3yAt$	بافعالاتها	$bAC_1C_2AC_3AthA$
بافعالياته	$bAC_1C_2AC_3yAth$	بالاستفعال	$bAlAstC_1C_2AC_3$
بالاستفعلة	$bAlAstC_1C_2C_3t$	بالافتعالي	$bAlAC_1tC_2AC_3y$
بالافعالات	$bAlAC_1C_2AC_3At$	بالتفاعلات	$bAltC_1AC_2C_3At$
بالتفعيلات	$bAltC_1C_2yC_3At$	بالفعالتين	$bAlC_1C_2AC_3tyn$
بالفعاليات	$bAlC_1C_2AC_3yAt$	بالفعلانية	$bAlC_1C_2C_3Anyt$
بالفعلاوات	$bAlC_1C_2C_3AwAt$	بالفعوليات	$bAlC_1C_2wC_3yAt$
بالفعوليين	$bAlC_1C_2wC_3yyn$	بالمتفاعلة	$bAlmtC_1AC_2C_3t$
بالمتفعلات	$bAlmtC_1C_2C_3At$	بالمتفعلين	$bAlmtC_1C_2C_3yn$
بالمفاعلات	$bAlmC_1AC_2C_3At$	بالمفاعلين	$bAlmC_1AC_2C_3yn$
بالمفتعلات	$bAlmC_1tC_2C_3At$	بالمفتعلين	$bAlmC_1tC_2C_3yn$
بالمفعليات	$bAlmC_1C_2C_3yAt$	بالمفعولات	$bAlmC_1C_2wC_3At$
بالمفعولية	$bAlmC_1C_2wC_3yt$	بالمفعولين	$bAlmC_1C_2wC_3yn$
بانفعالاته	$bAnC_1C_2AC_3Ath$	بتفاعلاتها	$btC_1AC_2C_3AthA$
بتفعيلاتها	$btC_1C_2yC_3AthA$	بمفاعلاتها	$bmC_1AC_2C_3AthA$
بمفتعلاتها	$bmC_1tC_2C_3AthA$	بمفعلياتهن	$bmC_1C_2C_3yAthn$
بمفعولاتنا	$bmC_1C_2wC_3AtnA$	بمفعوليتها	$bmC_1C_2wC_3ythA$
بمفعوليتهم	$bmC_1C_2wC_3ythm$	تستفعلينها	$tstC_1C_2C_3ynhA$
تفعيلاتهما	$tC_1C_2yC_3AthmA$	سيفاعلونها	$syC_1AC_2C_3wnhA$
فالاستفعال	$fAlAstC_1C_2AC_3$	فالافتعالة	$fAlAC_1tC_2AC_3t$
فالافعالات	$fAlAC_1C_2AC_3At$	فالانفعالة	$fAlAnC_1C_2AC_3t$
فالتفاعلات	$fAltC_1AC_2C_3At$	فالتفعيلات	$fAltC_1C_2yC_3At$
فالفعوليات	$fAlC_1C_2wC_3yAt$	فالمستفعلة	$fAlmstC_1C_2C_3t$
فالمفاعلات	$fAlmC_1AC_2C_3At$	فالمفاعلون	$fAlmC_1AC_2C_3wn$
فالمفعولات	$fAlmC_1C_2wC_3At$	فبالمفعلات	$fbAlmC_1C_2C_3At$
فسيفعلونها	$fsyC_1C_2C_3wnhA$	ففعلانيتنا	$fC_1C_2C_3AnytnA$
فليستفعلوا	$flystC_1C_2C_3wA$	فيعلاناتهم	$C_1yC_2C_3AnAthm$
كاستفعالها	$kAstC_1C_2AC_3hA$	كالاستفعال	$kAlAstC_1C_2AC_3$
كالاستفعلة	$kAlAstC_1C_2C_3t$	كالافعالات	$kAlAC_1C_2AC_3At$
كالتفعيلات	$kAltC_1C_2yC_3At$	كالفعاليات	$kAlC_1C_2AC_3yAt$
كالفعلانات	$kAlC_1C_2C_3AnAt$	كالفعلانية	$kAlC_1C_2C_3Anyt$
كالفعوليات	$kAlC_1C_2wC_3yAt$	كالمتفعلين	$kAlmtC_1C_2C_3yn$
كالمفاعلات	$kAlmC_1AC_2C_3At$	كالمفتعلات	$kAlmC_1tC_2C_3At$
كالمفعليين	$kAlmC_1C_2C_3yyn$	كالمفعولات	$kAlmC_1C_2wC_3At$
لاستفعالات	$lAstC_1C_2AC_3At$	لاستفعالها	$lAstC_1C_2AC_3hA$
لاستفعالهم	$lAstC_1C_2AC_3hm$	لاستفعالية	$lAstC_1C_2AC_3yt$
لاستفعلتها	$lAstC_1C_2C_3thA$	لافتعالاته	$lAC_1tC_2AC_3Ath$
لانفعالاته	$lAnC_1C_2AC_3Ath$	للاستفعلات	$llAstC_1C_2C_3At$
للافتعالات	$llAC_1tC_2AC_3At$	للافتعالية	$llAC_1tC_2AC_3yt$
للافعاليين	$llAC_1C_2AC_3yyn$	للانفعالات	$llAnC_1C_2AC_3At$
للفعليين	$llC_1C_2C_3ynyyn$	للمتافعلين	$llmtAC_1C_2C_3yn$

table continued on next page

Table A.7 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
للمستفعلات	*llmstC1C2C3At*	للمستفعلين	*llmstC1C2C3yn*
للمفتعليين	*llmC1tC2C3yyn*	للمفعوليات	*llmC1C2wC3yAt*
لمستفعليها	*lmstC1C2C3yhA*	لمفتعلاتها	*lmC1tC2C3AthA*
لمفعوليتها	*lmC1C2wC3ythA*	لمفعوليتهم	*lmC1C2wC3ythm*
ليستفعلنهم	*lystC1C2C3nhm*	ليفاعلونا	*lyC1AC2C3wnnA*
ليفاعلونهم	*lyC1AC2C3wnhm*	مستفعلاتها	*mstC1C2C3AthA*
مفاعيليتها	*mC1AC2yC3ythA*	مفتعلاتهم	*mC1tC2AC3Athm*
مفتعلاتهما	*mC1tC2C3AthmA*	مفعالياتهم	*mC1C2AC3yAthm*
مفعولياتها	*mC1C2wC3yAthA*	مفعولياتهم	*mC1C2wC3yAthm*
مفعوليتهما	*mC1C2wC3ythmA*	واستفعالات	*wAstC1C2AC3At*
واستفعالنا	*wAstC1C2AC3nA*	واستفعالها	*wAstC1C2AC3hA*
واستفعالهم	*wAstC1C2AC3hm*	واستفعاليا	*wAstC1C2AC3yA*
واستفعالية	*wAstC1C2AC3yt*	واستفعلتها	*wAstC1C2C3thA*
واستفعلتهم	*wAstC1C2C3thm*	وافتعالاته	*wAC1tC2AC3Ath*
وافتعاليات	*wAC1tC2AC3yAt*	وافتعاليته	*wAC1tC2AC3yth*
وافتعاليون	*wAC1tC2AC3ywn*	وافعالاتها	*wAC1C2AC3AthA*
وافعالاتهم	*wAC1C2AC3Athm*	وافعالياته	*wAC1C2AC3yAth*
وافعاليتهم	*wAC1C2AC3ythm*	والاستفعال	*wAlAstC1C2AC3*
والاستفعلة	*wAlAstC1C2C3t*	والافتعالى	*wAlAC1tC2AC3A*
والافتعالي	*wAlAC1tC2AC3y*	والافعالات	*wAlAC1C2AC3At*
والافعالية	*wAlAC1C2AC3yt*	والافعليات	*wAlAC1C2C3yAt*
والافعليين	*wAlAC1C2C3yyn*	والافعولات	*wAlAC1C2wC3At*
والافعولية	*wAlAC1C2wC3yt*	والافعيلية	*wAlAC1C2yC3yt*
والتفاعلات	*wAltC1AC2C3At*	والتفاعلية	*wAltC1AC2C3yt*
والتفعيلات	*wAltC1C2yC3At*	والتفعيلية	*wAltC1C2yC3yt*
والفاعليات	*wAlC1AC2C3yAt*	والفاعلين	*wAlC1AC2C3yyn*
والفاعليين	*wAlC1AC2C3yyn*	والفاعولات	*wAlC1AC2wC3At*
والفاعولية	*wAlC1AC2wC3yt*	والفاعيلات	*wAlC1AC2yC3At*
والفعاليتان	*wAlC1C2AC3tAn*	والفعاليات	*wAlC1C2AC3yAt*
والفعاليون	*wAlC1C2AC3ywn*	والفعاليين	*wAlC1C2AC3yyn*
والفعلاتية	*wAlC1C2C3Atyt*	والفعلانية	*wAlC1C2C3Anyt*
والفعلاوات	*wAlC1C2C3AwAt*	والفعليتان	*wAlC1C2C3ytAn*
والفعوليات	*wAlC1C2wC3yAt*	والفعوليون	*wAlC1C2wC3ywn*
والفعوليين	*wAlC1C2wC3yyn*	والفعيلتين	*wAlC1C2yC3tyn*
والفعيليات	*wAlC1C2yC3yAt*	والفعيليين	*wAlC1C2yC3yyn*
والفعولية	*wAlC1mC2wC3yt*	والفويعلات	*wAlC1wyC2C3At*
والمتفاعلة	*wAlmtC1AC2C3t*	والمتفعلات	*wAlmtC1C2C3At*
والمتفعلون	*wAlmtC1C2C3wn*	والمتفعلية	*wAlmtC1C2C3yt*
والمتفعلين	*wAlmtC1C2C3yn*	والمتمفعلة	*wAlmtmC1C2C3t*
والمستفعلة	*wAlmstC1C2C3t*	والمفاعلات	*wAlmC1AC2C3At*
والمفاعلون	*wAlmC1AC2C3wn*	والمفاعلين	*wAlmC1AC2C3yn*
والمفتعلات	*wAlmC1tC2C3At*	والمفتعلون	*wAlmC1tC2C3wn*
والمفتعلين	*wAlmC1tC2C3yn*	والمفعلاتي	*wAlmC1C2C3Aty*
والمفعليات	*wAlmC1C2C3yAt*	والمفعليان	*wAlmC1C2C3yAn*
والمفعليين	*wAlmC1C2C3yyn*	والمفعولات	*wAlmC1C2wC3At*
والمفعولون	*wAlmC1C2wC3wn*	والمفعولية	*wAlmC1C2wC3yt*

table continued on next page

Table A.7 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
والمفعولين	$wAlmC_1C_2wC_3yn$	والمنفعلات	$wAlmnC_1C_2C_3At$
والمنفعلين	$wAlmnC_1C_2C_3yn$	وانفعالاته	$wAnC_1C_2AC_3Ath$
وباستفعلتي	$wbAstC_1C_2C_3ty$	وبالافتعال	$wbAlAC_1tC_2AC_3$
وبالتفعلات	$wbAltC_1C_2C_3At$	وبالفاعلين	$wbAlC_1AC_2C_3yn$
وبالفعالات	$wbAlC_1C_2AC_3At$	وبالفعليات	$wbAlC_1C_2C_3yAt$
وبالفعولية	$wbAlC_1C_2wC_3yt$	وبالمستفعل	$wbAlmstC_1C_2C_3$
وبالمفاعلة	$wbAlmC_1AC_2C_3t$	وبالمفعلات	$wbAlmC_1C_2C_3At$
وبالمفعلين	$wbAlmC_1C_2C_3yn$	وبتفعيلاته	$wbtC_1C_2yC_3Ath$
وبفعلانهما	$wbC_1C_2C_3AnhmA$	وبمفاعلهما	$wbmC_1AC_2C_3hmA$
وبمفتعلاته	$wbmC_1tC_2C_3Ath$	وبمفعلاتهم	$wbmC_1C_2C_3Athm$
وتفاعلاتها	$wtC_1AC_2C_3AthA$	وتفعيلاتنا	$wtC_1C_2yC_3AtnA$
وتفعيلاتها	$wtC_1C_2yC_3AthA$	وتفعيلاتهم	$wtC_1C_2yC_3Athm$
وسيفعلوننا	$wsyC_1C_2C_3wnnA$	وفعالاتهما	$wC_1C_2AC_3AthmA$
وفعالياتها	$wC_1C_2AC_3yAthA$	وفعلانيتنا	$wC_1C_2C_3AnytnA$
وفعولياتهم	$wC_1C_2wC_3yAthm$	ولاستفعاله	$wlAstC_1C_2AC_3h$
ولافتعالات	$wlAC_1tC_2AC_3At$	ولانفعالها	$wlAnC_1C_2AC_3hA$
ولتستفعلوا	$wltstC_1C_2C_3wA$	وللاستفعلة	$wllAstC_1C_2C_3t$
وللمفاعلين	$wllmC_1AC_2C_3yn$	وللمفعولات	$wllmC_1C_2wC_3At$
ومتفعلاتها	$wmtC_1C_2C_3AthA$	ومتفعلاتهم	$wmtC_1C_2C_3Athm$
ومستفعليها	$wmstC_1C_2C_3yhA$	ومفاعلاتكم	$wmC_1AC_2C_3Atkm$
ومفاعلاتها	$wmC_1AC_2C_3AthA$	ومفتعلاتها	$wmC_1tC_2C_3AthA$
ومفتعلاتهم	$wmC_1tC_2C_3Athm$	ومفعولاتها	$wmC_1C_2wC_3AthA$
ومفعولاتيا	$wmC_1C_2wC_3AtyA$	ومفعوليتها	$wmC_1C_2wC_3ythA$
يتفاعلونها	$ytC_1AC_2C_3wnhA$	يستفعلونني	$ystC_1C_2C_3wnny$
يستفعلونها	$ystC_1C_2C_3wnhA$		

Table A.8: A List of Compiled Arabic Patterns for $patt_{11}$

Pattern	Pattern Form	Pattern	Pattern Form
استفعالاتكم	$AstC_1C_2AC_3Atkm$	استفعالاتنا	$AstC_1C_2AC_3AtnA$
استفعالاتها	$AstC_1C_2AC_3AthA$	استفعالاتهم	$AstC_1C_2AC_3Athm$
استفعالاتهن	$AstC_1C_2AC_3Athn$	افتعالاتهما	$AC_1tC_2AC_3AthmA$
افتعالياتها	$AC_1tC_2AC_3yAthA$	الاستفعالات	$AlAstC_1C_2AC_3At$
الاستفعالية	$AlAstC_1C_2AC_3yt$	الاستفعالين	$AlAstC_1C_2AC_3yn$
الاستفعليين	$AlAstC_1C_2C_3yyn$	الافتعاليات	$AlAC_1tC_2AC_3yAt$
الافتعاليون	$AlAC_1tC_2AC_3ywn$	الافتعاليين	$AlAC_1tC_2AC_3yyn$
الافعاليتين	$AlAC_1C_2AC_3ytyn$	الانفعاليين	$AlAnC_1C_2AC_3yyn$
الفاعلانيين	$AlC_1AC_2C_3Anyyn$	الفعالاتيون	$AlC_1C_2AC_3Atywn$
الفعالاتيين	$AlC_1C_2AC_3Atyyn$	الفعالينيات	$AlC_1C_2AC_3ynyAt$
المتفاعلتان	$AlmtC_1AC_2C_3tAn$	المتفاعلتين	$AlmtC_1AC_2C_3tyn$
المستفعلتين	$AlmstC_1C_2C_3tyn$	المستفعليات	$AlmstC_1C_2C_3yAt$
المفعولاتية	$AlmC_1C_2wC_3Atyt$	انفعالاتهما	$AnC_1C_2AC_3AthmA$
باستفعالاته	$bAstC_1C_2AC_3Ath$	بافتعالاتها	$bAC_1tC_2AC_3AthA$

table continued on next page

Table A.8 – *continued from previous page*

Pattern	Pattern Form	Pattern	Pattern Form
بافتعاليتها	$bAC_1tC_2AC_3ythA$	بالافتعالات	$bAlAC_1tC_2AC_3At$
بالافتعالية	$bAlAC_1tC_2AC_3yt$	بالافتعالين	$bAlAC_1tC_2AC_3yn$
بالافعاليات	$bAlAC_1C_2AC_3yAt$	بالافعاليين	$bAlAC_1C_2AC_3yyn$
بالافعيالات	$bAlAC_1C_2yAC_3At$	بالانفعالات	$bAlAnC_1C_2AC_3At$
بالانفعالية	$bAlAnC_1C_2AC_3yt$	بالفعلانيات	$bAlC_1C_2C_3AnyAt$
بالفعلانيين	$bAlC_1C_2C_3Anyyn$	بالمستفعلات	$bAlmstC_1C_2C_3At$
بالمستفعلين	$bAlmstC_1C_2C_3yn$	بالمفعلانات	$bAlmC_1C_2C_3AnAt$
بالمفعوليات	$bAlmC_1C_2wC_3yAt$	بالمفعوليين	$bAlmC_1C_2wC_3yyn$
بتفاعلياتها	$btC_1AC_2C_3yAthA$	بمستفعلاتها	$bmstC_1C_2C_3AthA$
سيستفعلونها	$systC_1C_2C_3wnhA$	فالافتعالات	$fAlAC_1tC_2AC_3At$
فالافتعالية	$fAlAC_1tC_2AC_3yt$	فالانفعالات	$fAlAnC_1C_2AC_3At$
فالمستفعلات	$fAlmstC_1C_2C_3At$	فالمستفعلون	$fAlmstC_1C_2C_3wn$
فالمفعوليون	$fAlmC_1C_2wC_3ywn$	كالافتعالية	$kAlAC_1tC_2AC_3yt$
كالمستفعلات	$kAlmstC_1C_2C_3At$	كالمفعوليات	$kAlmC_1C_2wC_3yAt$
كالمفعوليين	$kAlmC_1C_2wC_3yyn$	لافتعالاتها	$lAC_1tC_2AC_3AthA$
لافتعالاتهم	$lAC_1tC_2AC_3Athm$	لافعالياتها	$lAC_1C_2AC_3yAthA$
للاستفعالات	$llAstC_1C_2AC_3At$	للاستفعالية	$llAstC_1C_2AC_3yt$
للافتعاليات	$llAC_1tC_2AC_3yAt$	للافتعاليين	$llAC_1tC_2AC_3yyn$
للمفعولاتية	$llmC_1C_2wC_3Atyt$	واستفعالاته	$wAstC_1C_2AC_3Ath$
واستفعالهما	$wAstC_1C_2AC_3hmA$	واستفعاليين	$wAstC_1C_2AC_3yyn$
وافتعالاتها	$wAC_1tC_2AC_3AthA$	وافتعالاتهم	$wAC_1tC_2AC_3Athm$
وافعالياتها	$wAC_1C_2AC_3yAthA$	وافعالياتهم	$wAC_1C_2AC_3yAthm$
وافعلتموهما	$wAC_1C_2C_3tmwhmA$	والاستفعالي	$wAlAstC_1C_2AC_3y$
والاستفعلات	$wAlAstC_1C_2C_3At$	والافتعالات	$wAlAC_1tC_2AC_3At$
والافتعالية	$wAlAC_1tC_2AC_3yt$	والافعاليات	$wAlAC_1C_2AC_3yAt$
والافعاليون	$wAlAC_1C_2AC_3ywn$	والافعاليين	$wAlAC_1C_2AC_3yyn$
والافعيالية	$wAlAC_1C_2yAC_3yt$	والانفعالات	$wAlAnC_1C_2AC_3At$
والتفعيليات	$wAltC_1C_2yC_3yAt$	والتفعيليين	$wAltC_1C_2yC_3yyn$
والفعالينات	$wAlC_1C_2AC_3ynAt$	والفعلانيون	$wAlC_1C_2C_3Anywn$
والفعلانيين	$wAlC_1C_2C_3Anyyn$	والمتفاعلين	$wAlmtC_1AC_2C_3yn$
والمستفعلات	$wAlmstC_1C_2C_3At$	والمستفعلون	$wAlmstC_1C_2C_3wn$
والمستفعلية	$wAlmstC_1C_2C_3yt$	والمستفعلين	$wAlmstC_1C_2C_3yn$
والمفعوليون	$wAlmC_1C_2wC_3ywn$	وانفعالاتنا	$wAnC_1C_2AC_3AtnA$
وانفعالاتها	$wAnC_1C_2AC_3AthA$	وبالاستفعال	$wbAlAstC_1C_2AC_3$
وبالاستفعلة	$wbAlAstC_1C_2C_3t$	وبالفعالتين	$wbAlC_1C_2AC_3tyn$
وبالفعوليات	$wbAlC_1C_2wC_3yAt$	وبالمفاعلات	$wbAlmC_1AC_2C_3At$
وبالمفاعلات	$wbAlmC_1AC_2C_3At$	وبالمفعولات	$wbAlmC_1C_2wC_3At$
وبمفتعلاتنا	$wbmC_1tC_2C_3AtnA$	وللمتفاعلين	$wllmtC_1AC_2C_3yn$
ومفعولياتها	$wmC_1C_2wC_3yAthA$		

Table A.9: A List of Compiled Arabic Patterns for $patt_{12}$

Pattern	Pattern Form	Pattern	Pattern Form
الاستفعاليون	$AlAstC_1C_2AC_3ywn$	باستفعالاتها	$bAstC_1C_2AC_3AthA$
بالافتعاليات	$bAlAC_1tC_2AC_3yAt$	بالمستفعليات	$bAlmstC_1C_2C_3yAt$
بالمفعولاتية	$bAlmC_1C_2wC_3Atyt$	فاستفعلتماها	$fAstC_1C_2C_3tmAhA$
فاستفعلتموها	$fAstC_1C_2C_3tmwhA$	فاستفعلتموهم	$fAstC_1C_2C_3tmwhm$
فالاستفعالات	$fAlAstC_1C_2AC_3At$	فالافتعاليات	$fAlAC_1tC_2AC_3yAt$
فبالمستفعلات	$fbAlmstC_1C_2C_3At$	فبالمفعوليات	$fbAlmC_1C_2wC_3yAt$
كالمفعولاتية	$kAlmC_1C_2wC_3Atyt$	لاستفعالاتنا	$lAstC_1C_2AC_3AtnA$
لاستفعالاتنا	$lAstC_1C_2AC_3AtnA$	لاستفعالاتها	$lAstC_1C_2AC_3AthA$
واستفعالاتكم	$wAstC_1C_2AC_3Atkm$	واستفعالاتهم	$wAstC_1C_2AC_3Athm$
واستفعلتماها	$wAstC_1C_2C_3tmAhA$	واستفعلتموها	$wAstC_1C_2C_3tmwhA$
واستفعلتموهم	$wAstC_1C_2C_3tmwhm$	وافتعالياتها	$wAC_1tC_2AC_3yAthA$
وافتعالياتها	$wAC_1tC_2AC_3yAthA$	والاستفعالات	$wAlAstC_1C_2AC_3At$
والاستفعالية	$wAlAstC_1C_2AC_3yt$	والافتعاليات	$wAlAC_1tC_2AC_3yAt$
والافتعاليون	$wAlAC_1tC_2AC_3ywn$	والافتعاليين	$wAlAC_1tC_2AC_3yyn$
والفعالينيات	$wAlC_1C_2AC_3ynyAt$	واللامفعولية	$wAllAmC_1C_2wC_3yt$
والمفعولاتية	$wAlmC_1C_2wC_3Atyt$	وبالمفعوليات	$wbAlmC_1C_2wC_3yAt$

Appendix B

The Complete List of the Compiled Arabic Affixes

B.1 The Compiled List of Affix letters

Table B.1: The Affixes of $patt_5$

Prefixes			Infixes			Suffixes		
ال	ات	ا	ي	و	ا	ا	اء	ات
ت	ب	ان				اك	ان	اه
تن	تم	تت				ة	تا	تك
ل	ك	ف				تم	تن	ته
م	لل	لت				ك	كم	كن
ن	من	مت				ن	نا	نك
و	نن	نت				نه	ني	ه
ي	وي	وت				ها	هم	هن
	ين	يت				وا	وك	وه
						ى	ي	يا
						ية	ين	

Table B.2: The Affixes of $patt_6$

Prefixes			Infixes			Suffixes		
استـ	اتـ	اـ	ي	وـ	اـ	ا	اء	ات
الم	الت	ال		أ	ت	اتي	اكم	اكن

table continued on next page

Table B.2 – *continued from previous page*

Prefixes			Infixes	Suffixes		
بال	ب	ان		ال	ان	انا
ت	بم	بت		اني	اه	اها
تن	تست	تت		اهم	اهن	ة
ل	كال	فتت		تا	تاك	تاه
لي	للت	لل		تكم	تكن	تم
مست	مت	م		تما	تن	تنا
نست	نت	ن		تني	تها	تهم
وال	وا	و		تهن	تي	كم
ي	وم	وت		كما	كن	ن
ين	يست	يت		نا	ناك	ناه
				نكم	نكن	ننا
				نه	ننني	نها
				نهم	نهن	ني
				ه	ها	هم
				هما	هن	وا
				وكم	وكن	ون
				ونا	وني	وه
				وها	وهم	وهن
				ى	ي	يات
				ية	ين	يه

Table B.3: The Affixes of $patt_7$

Prefixes			Infixes			Suffixes		
است	اتست	ا	ا	و	ي	ا	اء	ات
الت	الا	ال	ت	أ		اتي	اكما	ان
ان	المت	الم				انه	اها	اهم
بال	با	اي				اهما	اهن	ة
ت	بم	بالم				تا	تاكم	تاكن
تن	تست	تت				تان	تانا	تاني
ل	كم	كالم				تاها	تاهم	تاهن
لم	للت	لت				تكما	تم	تما
ن	مست	م				تموه	تن	تنا
واست	وا	و				ته	تهما	توها
وان	والم	وال				توهم	تي	تين
ي	وتت	وت				كما	ن	نا
ين	يست	يت				ناكم	ناكن	نانا
						ناني	ناها	ناهم
						نكما	ناهن	نها
						نهما	نهم	نهن
						ه	ني	ها
						هما	هم	وا
						ون	وكما	ونه
						وهم	وها	وهما
						ي	وهن	يات

table continued on next page

Table B.3 – *continued from previous page*

Prefixes	Infixes	Suffixes		
		ية	يتها	ين
		يها	يهم	يهن

Table B.4: The Affixes of $patt_8$

Prefixes			Infixes			Suffixes		
ا	ات	است	ي	و	ا	ا	اء	اننا
ال	الا	الان		أ	ت	ات	اته	اتها
الت	الم	المت				اتهم	اتهن	اتي
المن	ان	ب				ان	انها	انهم
با	باست	بال				انهن	اهما	ة
بالا	بالم	بت				تا	تاكما	تاهما
بم	ت	تت				تك	تم	تما
تست	فالت	ك				تماها	تموها	تموهن
كالم	لاست	لل				تن	تني	ته
للا	للت	لم				تها	تهم	تين
م	و	وا				كما	نا	ناكما
وال	والا	والت				ناهما	نهما	ني
والم	وت	وم				ه	ها	هم
ومست	ي	يت				هما	هن	وا
يست						ون	ونا	وننا
						ونهما	ونهم	ونهن
						وني	وه	وها
						وهما	ي	يا
						يات	ية	يته
						ين	ين	يني
						ينيات	يه	يها
						يهما	يون	يين

Table B.5: The Affixes of $patt_9$

Prefixes			Infixes			Suffixes		
أ	إ	إست	ي	و	ا	ا	اء	اءنا
ا	اس	است	ل	ف	ت	انها	انهما	الية
ال	الأ	الإ			م	ات	اتك	اتكم
الأست	الإست	الا				اتنا	اته	اتها
الاست	الان	الت				اتهم	اتهما	اتهن
اللا	اللات	اللام				اتي	اتية	اك
الم	المت	المتم				الي	ان	انات
المست	المستي	المن				انة	انها	انهما
ان	ب	بأ				اني	انيات	انية
بإ	با	باست				انيته	انيتها	انيين

table continued on next page

Table B.5 – *continued from previous page*

Suffixes			Infixes	Prefixes		
ة	اوية	اوي		بالإ	بالأ	بال
تك	تاهما	تان		بالت	بالان	بالا
تما	تم	تكم		بان	بالمست	بالم
تنا	تموهما	تموها		بمست	بم	بت
تها	ته	تني		سا	تست	ت
تهن	تهما	تهم		سي	سنت	ستست
ك	تين	تي		ف	سيست	سيت
نا	ن	مائة		فال	فاست	فا
نها	ناها	ناه		فالان	فالا	فالأ
نيون	ني	نهم		فبالإ	فالم	فالت
هم	ها	ه		فللم	فتست	فبالم
وا	هن	هما		فيت	في	فم
وكم	وانين	وات		كا	ك	فيست
ونكم	ونك	ون		كالان	كالا	كال
ونه	وننني	وننا		كالمست	كالمت	كالم
ونهما	ونهم	ونها		لأ	ل	كمست
وهم	وها	وه		لا	لإست	لإ
ى	ويون	وهما		لامت	لاست	لاا
يات	يا	ي		لل	لت	لان
ياته	ياتنا	ياتك		للا	للإست	للإ
ياتي	ياتهم	ياتها		للم	للت	للاست
يتنا	يتان	ية		لم	للمست	للمت
يتهم	يتها	يته		ليت	لي	لمست
يل	يتين	يتهما		مت	م	ليست
ينات	ينا	ين		و	من	مست
ينية	ينيات	يننني		واست	وإ	وأ
يهم	يها	يه		وال	واست	وا
ييها	يين	يون		والا	والإ	والأ
				والم	والت	والان
				والمست	والمتم	والمت
				واليا	والمو	والمن
				وبإ	وب	وان
				وبال	وباست	وبا
				وبالت	وبالا	وبالأ
				وبم	وبت	وبالم
				وكا	وكإ	وت
				ولأ	ول	وكال
				ولل	ولت	ولإ
				وللم	وللت	وللا
				وم	ولن	ولم
				ومن	ومست	ومت
				ويست	ويت	وي
				يست	يت	ي

Table B.6: The Affixes of $patt_{10}$

Prefixes			Infixes			Suffixes		
ا	است	ال	ي	و	ا	ات	اتكم	اتنا
الا	الاست	الان	ل	ف	ت	اته	اتها	اتهم
الت	الم	المت			م	اتهما	اتي	اتيا
المست	المن	ان				اتية	اتين	ان
با	باست	بال				انات	اناتهم	انهما
بالا	بالاست	بالت				انية	انيات	انيتنا
بالم	بالمت	بان				انيون	انيين	اوات
بت	بم	ت				ة	تان	تها
تست	سي	ف				تهم	تي	تين
فال	فالا	فالاست				نا	ناها	نهم
فالان	فالت	فالم				ه	ها	هم
فالمست	فبالم	فسي				هما	وا	وانية
فليست	كاست	كال				ون	وننا	وننني
كالا	كالاست	كالت				ونها	ونهم	ى
كالم	كالمت	لا				ي	يا	يات
لاست	لان	لل				ياته	ياتها	ياتهم
للا	للاست	للان				ياتهن	يان	ية
للم	للمست	لم				يتان	يته	يتها
لمست	لي	ليست				يتهم	يتهما	يتين
م	مست	و				ين	ينات	ينها
وا	واست	وال				ينيون	ينيون	ينيين
والا	والاست	والت					يون	يين
والم	والمت	والمتم						
والمست	والمن	وان						
وب	وباست	وبال						
وبالا	وبالت	وبالم						
وبالمست	وبت	وبم						
وت	وسي	ولا						
ولاست	ولان	ولتست						
وللاست	وللم	ولم						
ومت	ومست	يت						
يست								

Table B.7: The Affixes of $patt_{11}$

Prefixes			Infixes			Suffixes		
ا	است	ال	ي	و	ا	ات	اتكم	اتنا
الا	الاست	الان			ت	اته	اتها	اتهم
الم	المت	المست				اتهما	اتهن	اتية
ان	با	باست				اتيون	اتيين	انات
بال	بالا	بالان				انيات	انيون	انيين
بالم	بالمست	بت				ة	تان	تموهما
بمست	سيست	فالا				تين	هما	ون
فالان	فالم	فالمست				ونها	ي	يات
كالا	كالم	كالمست				ياتها	ياتهم	ية

table continued on next page

Table B.7 – *continued from previous page*

Prefixes			Infixes	Suffixes		
لا	للا	للاست		يتها	يتين	ين
للم	وا	واست		ينات	ينيات	يون
وال	والا	والاست		يين		
والان	والت	والم				
والمت	والمست	وان				
وبال	وبالاست	وبالم				
وبم	وللمت	وم				

Table B.8: The Affixes of $patt_{12}$

Prefixes			Infixes			Suffixes		
الاست	باست	بالا	ا	و	ي	ات	اتكم	اتنا
بالم	بالمست	فاست	ت			اتها	اتهم	اتية
فالا	فالاست	فبالم				تماها	تموها	تموهم
فبالمست	كالم	لاست				يات	ياتها	ية
وا	واست	وال				ينيات	يون	يين
والا	والاست	واللام						
والم	وبالم							

B.2 The Compiled Prefix-Suffix Composition Lists

Table B.9: The Prefix-Suffix Composition of $patt_5$

Prefix	Suffixes
ا	null ا ة ك ن ه ي
ات	null
ال	null
ان	null
ب	null ة ه
ت	null ا ة ك ن ه ي
تت	null
تم	null
تن	null
سا	null
ست	null
سن	null
ف	ة ه
فب	null
ك	null
ل	ة ه
لت	null
لل	null
م	null ا ة و ي
مت	null
من	null
ن	null ك ه
نت	null
نن	null
و	null ة ه
وت	null
وي	null
ي	null ا ك ن ه
يت	null
ين	null

Table B.10: The Prefix-Suffix Composition of $patt_6$

Prefix	Suffixes
ا	null ا اء اه ت تا تم تن تكم تكن نا نه ني ه ها هم هن وا وه يه ي
ات	null
است	null

Continued on next page

Table B.10 – continued from previous page

Prefix	Suffixes
ال	null ة ى ي
الت	null
الم	null
ان	null ا ت ن ي
ب	null ة ه
بال	null
بت	null
بم	null ه
ت	null ا ات ان ت تا تم تن تم كن كم ن نا نه ني ه ها هم هن وا ون ي ين
تت	null ا ن ي
تست	null
تن	ا ن ي
سا	ه
ست	ه
سن	ه
فتت	null
كال	null
ل	ة
لل	null
للت	null
لي	ن
م	null ات ان ة تي ه وا ون ين
مت	null
مست	null
ن	كم ها كن هم هن
نت	null
نست	null
و	اء ة ها هم
وا	null
وال	null
وت	null
وسا	null
وست	null
وسن	null
وم	null
ي	ا ان كم كن ن نا نه ني ها هم هن وا ون
يت	null ا ن
يست	null
ين	ا ن

Table B.11: The Prefix-Suffix Composition of $patt_7$

Prefix	Suffixes
ا	ات اها اهم اهن تما نهاكما نهم نهن ه ها هما وها وهم وهن ي ية يها يهم يهن
اتست	null
است	null ا ت ك ن
ال	null اء ات ان ة ي ية ين
الا	null
الت	null
الم	null ة ه ي
المت	null
ان	تا تم تن نا وا
اي	ني
با	null
بال	null
بالم	null
بم	تي ة
ت	ان انه تا تم تن تما ناكما نها نهم نهن ها هما وا ون ونه ين ينه
تت	ا ن وا ي
تست	ا ن ي
تن	ان ون وا ين
سا	هم
كالم	null
كم	تي
ل	ته
لت	ان
للت	null
لم	ة
م	ات اتي ان اني تان ته هم ون
مست	ة
ن	كما هما
و	هم
وا	null
واست	null
وال	null
والم	null
وان	null
وت	null
وتت	null
ي	ان انه تما نهاكما نهم نهن ها هما وا ون ونه
يت	ا ن وا ون
يست	ا
ين	ان وا ون

Table B.12: The Prefix-Suffix Composition of $patt_8$

Prefix	Suffixes
ا	وهما وهاوني ونا هما هم هانهما ناكما تهم تها تنياهما ان اتياتهم اته ات
	ييين يونيهما ين ية يات يا
ات	ها ان
است	يا ي وه وا هم ها ه ني نا ته تن تم تا ات ا
ال	ييين يني ين ية يات ون تين ان ات null
الا	ية ي ات اء null
الان	null
الت	ي ات
الم	ين ية ون ة ان ات null
المت	ة
المن	ة
ان	تما
ب	يتها يته يات هما تها اته ائنا
با	ي هن هم ها ه نا تهم ة ات
باست	null
بال	ية ة null
بالا	null
بالم	ة null
بت	ها تها ات
بم	ية ها ات
ت	ونهزونهم ونها هما هم نهما تماانهن انهم انها
تت	ين ون وا ان
تست	ين ون وا ان
فالت	null
ك	يته
كالم	ة
لاست	null
لل	ين يات ات
للا	null
للت	ات
لم	تك
م	يات هماتهن اتهم
و	يها يته
وا	تهم
وال	ة اء null
والا	ة
والت	null
والم	null
وت	ات
وم	يه ات
ومست	ا
وي	ون
ي	ونهزونهم ونها وننا نهما انهن انهم انها
يت	ون وا ان
يست	ون وا ني ان

Table B.13: The Prefix-Suffix Composition of $patt_9$

Prefix	Suffixes
أ	ائهما اتهااتك
إ	يتين يتان ياتياتك اتهن اتهم اتها اتنا
إست	هما تها
ا	يين يون يهلينيات يتين يتها ياته يات هما ناها تهما اتي اتهم اتها اته اتنااتك ائهما
اس	اته
است	ية يا وهلوكم هما هم ها ناه نا تهم تها تني تنا تما تكم ات
ال	يين يون ينية ينات يتين يتان ية ياتويونيون تين تان اوية انية اني انات اتي ائية
الأ	ين ة ات
الإ	ية ات
الأست	null
الإست	null ة
الا	يين يون ين ية ي ى ة ان اتي ات
الاست	null ي ة
الان	ي ة
الت	ين ية ات
اللا	ي ة
اللات	null
اللام	null ي
الم	يين يون ين ية يات ون تين تان ة اني ان اتي ات
المت	ين ية ون ة ان ات
المتم	ة
المست	ي ة
المستي	null
المن	ون ات
ان	تها اتي اته
ب	يتهلياتهم ياتها ياته تهما اتهم اتها اتنا
بأ	ياته يات
بإ	يات
با	ية يات هما هم ها نا اتها ات
باست	ه ك تي ته
بال	يين ين ية يات وات ة ان ات null
بالأ	null ة اء
بالإ	null ة
بالا	null ة ية
بالان	null
بالت	null ية ات
بالم	null ين ية ن ة ان ات
بالمست	null
بان	ها ات
بت	يتها هما ها اته
بم	يتها يته يات ها تين تهن تهم تها تنا اتي اتها اته
بمست	هم ات
ت	يننني يتها ياتونهما ونهاانهما اتهم اتها اتنا اتكم
تست	ونه
سا	هما
ستست	ين

Continued on next page

Table B.13 – continued from previous page

Prefix	Suffixes
سنت	ها
سي	ونها وننا
سيت	ون ان
سيست	ون ها
ف	يتهم
فا	هم تها اته
فاست	ه
فال	null ين ية يات ون ان ات
فالأ	اء ة
فالا	null اء
فالان	null
فالت	null
فالم	null ية ون ة ان ات
فبالإ	null
فبالم	null
فتست	نا وا
فللم	ين
فم	وها
في	ونها وها
فيت	ون
فيست	ون ها
ك	وانين
كا	نا
كال	ات ين ية
كالا	null
كالان	null
كالم	null ين ة ان ات
كالمت	ة
كالمست	null
كمست	ين ات
ل	يتها ياتنا ها اتها اتنا
لأ	هما اتنا
لإ	يات اته
لإست	ا
لا	ية يات هما هم ها ناه نا ات
لاا	ية
لاست	null ه ا
لامت	ة
لان	ات
لت	هما اته اتنا
لل	يين يتين يات تين انية انات
للإ	ية ات
للإست	null
للا	ات
للاست	null
للت	ات
للم	ين ية تين اتي ات
للمت	ين ات

Continued on next page

Table B.13 – continued from previous page

Prefix	Suffixes
للمست	ة
لم	ات ياتي تهم تها تنا اتها اته
لمست	ات
لي	وهما
ليت	وا
ليست	وا ها
م	اتنا يتهم يتها يتنا ياته ياتنا تهما اتية اتهن اتهما اتهم اتها
مت	اتها تين تان اتهم
مست	تين يهم ينا
من	اتهم
و	اتنا يتهم ييها يتهلياتهم ياتها ياته هم ها نا مائة تهما تهلموهاتاهمانيين انيته انيات اتية اتهم اتها
وا	اءنا يون يات هما هم نا اته
وإ	اتك يين اته
واست	ها
وا	ائها ات اتها اتهم اتك ته تها مائة نا ها هم هن وها يا يات ية يون يته
واست	ا ات تم ته نا ه ها هم وا وه ي
وال	null ات اتي الي ان انة اوي اني ة وات تين ون ي يات ية يل ين يون يين
والأ	null ي اء
والإ	ي
والا	null ات اء ك اة ي ية
والان	null
والت	null ات ة ي ية
والم	null ات ة ن ون ي ية ين
والمت	null ن ة
والمتم	null
والمست	null
والمن	ة
والمو	null
واليا	null
وان	ات ها ية
وب	اتهم تها تهم يتها
وبإ	ية هم
وبا	تها ه هم ية
وباست	null
وبال	null ات ان ة ية
وبالأ	null
وبالا	null
وبالت	null
وبالم	null ة
وبت	تة نا ها
وبم	ات ية
وت	اتك اته اتهم نا ها هم هما
وكإ	ات
وكا	ات
وكال	ات
ول	اتها تها ها
ولأ	نهم

Continued on next page

Table B.13 – continued from previous page

Prefix	Suffixes
ولإ	نا
ولت	هم ها
ولل	ين ات
وللا	null
وللت	ات
وللم	ين ة ات
ولم	يه تها
ولن	نهم
وم	يها وها هم تهم تها انات اتي اتهم اتها اته اتنا
ومت	يين اته
ومست	ين هم ات
ومن	اته
وي	وهمونهم ونها ونه انها
ويت	ونه ون ان
ويست	ون وا ها ني ان
ي	ونهماونهم ونهاونني وننا ونكم انهما
يت	ونها ونونك هما
يست	ونه نها

Table B.14: The Prefix-Suffix Composition of $patt_{10}$

Prefix	Suffixes
ا	يتها ياتهم ياتها ياته اتهم اتها اتنا اتكم
است	هما ناها اتهم اته
ال	يين ينيين ينيون ينيات ينات يتين ية وانية انيين انيون انية انيات
الا	يين يون ين ية يات ات
الاست	ية ي ات
الان	ية ات
الت	يين يون ية يان يات
الم	يين يون ية يات تين تان انية انات اتين اتي ات
المت	ين ون ان ات
المست	ين ية ون ان ات
المن	تان
ان	اتهم اتها
با	ياته يات هما اتها اته
باست	ية ها نا تهم تها ات
بال	يين يات تين اوات انية
بالا	ي ات
بالاست	ة null
بالت	ات
بالم	ين ية يات ات
بالمت	ين ة ات
بان	اته
بت	اتها

Continued on next page

Table B.14 – continued from previous page

Prefix	Suffixes
بم	اتنا اتها يتها ياتهن يتهم
ت	اتهما
تست	ينها
سي	ونها
ف	انيتنا
فال	يات
فالا	ات ة
فالاست	null
فالان	ة
فالت	ات
فالم	ات ون
فالمست	ة
فبالم	ات
فسي	ونها
فليست	وا
كاست	ها
كال	انات انية يات
كالا	ات
كالاست	null ة
كالت	ات
كالم	ات يين
كالمت	ين
لا	اته
لاست	ات تها ها هم ية
لان	اته
للا	يين ية ات
للاست	ات
للان	ات
للم	يات يين
للمست	ات ين
لم	اتها يتها يتهم
لمست	يها
لي	وننا ونهم
ليست	نهم
م	اتهم اتهما ياتها ياتهم يتها يتهما
مست	اتها
و	اتهما انيتنا ياتها ياتهم
وا	اته ات اتها اتهم يات ياته يته يتهم يون
واست	ات تها نا تهم ها هم يا ية
وال	ات اتية انية اوات تان تين يات ية يتان يون يين
والا	ات ى ي يات ية يين
والاست	null ة
والت	ات ية
والم	ات اتي ة ون يات يان ية ين يين
والمت	ات ة ون ية ين
والمتم	ة
والمست	ة

Continued on next page

Table B.14 – continued from previous page

Prefix	Suffixes
والمن	ات ين
وان	اته
وب	انهما
وباست	تي
وبال	ات ات يات ية ين
وبالا	null
وبالت	ات
وبالم	ات ة ين
وبالمست	null
وبت	اته
وبم	اته اتهم هما
وت	اتنا اتها اتهم
وسي	ونا
ولا	ات
ولاست	ه
ولان	ها
ولتست	وا
وللاست	ة
وللم	ات ين
وم	اتكم اتها اتهم اتيا يتها
ومت	اتها اتهم
ومست	يها
يت	ونها
يست	ونني ونها

Table B.15: The Prefix-Suffix Composition of $patt_{11}$

Prefix	Suffixes
ا	اتهما ياتها
است	اتكم اتنا اتها اتهم اتهن
ال	اتيون اتيين انيين ينيات
الا	ات يتين يون يين
الاست	ات ية ين يين
الان	يين
الم	اتية
المت	تان تين
المست	تين يات
ان	اتهما
با	اتها يتها
باست	اته
بال	انيات انيين
بالا	ات ية يات ين يين
بالان	ات ية
بالم	انات يات يين

Continued on next page

Table B.15 – continued from previous page

Prefix	Suffixes
بالمست	ات ين
بت	ياتها
بمست	اتها
سيست	ونها
فالا	ات ية
فالان	ات
فالم	يون
فالمست	ات ون
كالا	ية
كالم	يات يين
كالمست	ات
لا	اتها اتهم ياتها
للا	يات يين
للاست	ات ية
للم	اتية
وا	اتها اتهم ياتهم
واست	اته هما يين
وال	انيون ينات
والا	ات يات ية يون يين
والاست	ات ي
والان	ات
والت	يات يين
والم	يون
والمت	ين
والمست	ات ون ية ين
وان	اتنا اتها
وبال	تين يات
وبالاست	null ة
وبالم	ات
وبم	اتنا
وللمت	ين
وم	ياتها

Table B.16: The Prefix-Suffix Composition of $patt_{12}$

Prefix	Suffixes
الاست	يون
باست	اتها
بالا	يات
بالم	اتية
بالمست	يات
فاست	تماها تموهم تموها
فالا	يات
فالاست	ات

Continued on next page

Table B.16 – continued from previous page

Prefix	Suffixes
فبالم	يات
فبالمست	ات
كالم	اتية
لاست	اتنا اتها
وا	ياتها
واست	اتكم اتهم تماها تموها تموهم
وال	ينيات
والا	يات يون يين
والاست	ات ية
واللام	ية
والم	اتية
وبالم	يات

Appendix C

A Stoplist for Arabic Language

Table C.1: The Complete List of the Compiled Stopwords

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
الأخرى	أبدا	أحد	أحد	إحدى	أحمد	أحيانا
آخر	أخرى	أخرى	أخيرا	إطلاقا	أقل	أكثر
آل	ألا	إلا	ألست	ألستم	ألستن	ألسن
أليس	أليست	أليسوا	إليه	أما	إما	أمام
أمامك	أمامكم	أمامكما	أمامكن	أمامنا	أمامه	أمامها
أنه	إنه	أنها	إنها	أنا	إنا	إنك
أنهما	إنهم	أنكما	إنكما	إنما	أننا	إننا
إنهما	أني	إني	أهنا	أواخر	أيا	آية
أينما	أين	أيها	أيهم	أيهما	أيهن	أبو
أيضا	اللأنا	اللاتي	أخرى	أخيه	أذا	أشياء
أمامه	الأمر	الأن	الا	الان	التي	الجاري
الأمام	الذى	الذي	الذين	الرغم	السابق	السواء
الحالي	القائم	القول	اللا	اللتان	اللذان	اللذين
الغير	الله	الله	اللواتي	المدة	المزيد	المقبل
اللن	المنصرم	النحو	الى	اليا	اليد	اما
الممكن	انتم	انما	انه	أننا	أهمها	أياهم
ان	بإحدى	بآخر	بأقل	بأقل	بأمامك	بأمامكم
ايضا	بأمامكن	بأمامنا	بإمامهم	بأمامي	بإمكان	بأواخر
بأولئك	باي	باية	بأيها	بأيهم	بأيهما	بأيهن
باشياء	باضبط	باقل	بالإضافة	بالأمام	بالأمر	بالتاكيد
بالتي	بالذي	بالذين	بالرغم	بالغير	ببضعه	باللاتي
باللتان	باللتين	باللذان	باللذين	باللواتي	بالنسنه	بامام
بامامه	بامامها	بامامهن	بأيهم	ببضع	ببضعة	ببعض
بتلكهم	بتيك	بتلك	بحيث	بدون	بدوننا	بدونه

Continued on next page

Table C.1 – continued from previous page

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
بذي	بذلك	بذاك	بذا	بدونهن	بدونهما	بدونهم
بشيئان	بشيئا	بشيء	بشان	بسبب	برغم	بذينك
بغير	بعض	بعدم	بعدة	بعدئذ	بعد	بشيئين
بغيرها	بغيره	بغيرنا	بغيركن	بغيركما	بغيركم	بغيرك
بقولكا	بقولا	بقول	بغيري	بغيرهن	بغيرهما	بغيرهم
بكل	بقولهما	بقولها	بقوله	بقولنا	بقولكما	بقولكم
بنا	بن	بماذا	بما	بلى	بكما	بكم
بهتان	بهاتين	بهاتان	بها	بهؤلاء	بنسبة	بنحو
بهن	بهما	بهم	بهذين	بهذه	بهذان	بهتين
بيننا	بينما	بينكن	بينكما	بينكم	بينك	بين
تالية	بيني	بينهن	بينهما	بينهم	بينها	بينه
تقولان	تقول	تقل	تقريبا	تحت	تبين	تاليه
تلك	تكونون	تكونوا	تكونان	تكون	تقولون	تقولوا
حول	حتى	حاليا	حاشا	جو	جهه	جدا
حولهن	حولهما	حولهم	حولنا	حولكن	حولكم	حولك
حينذاك	حينا	حينئذ	حين	حيثما	حيث	حولي
دوننا	دون	دائما	خلال	حينها	حينه	حينما
ذاك	ذات	ذات	دونهن	دونهما	دونهم	دونه
رغما	رغم	ربما	ذي	ذو	ذلك	ذلك
سوف	سواءا	سواء	ستكونون	ستكونوا	ستكونان	ستكون
شيئين	شيئان	شيئا	شيء	سيكونون	سيكونان	سيكون
ضدها	ضده	ضدنا	ضدكن	ضدكما	ضدكم	ضدك
ضروري	ضرورة	ضدين	ضدي	ضدهنا	ضدهما	ضدهم
عدم	عدة	عدا	عامة	طالما	ضمن	ضروريا
عليه	علينا	عليكن	عليكما	عليكم	عليك	على
عند	عنا	عـــن	عليهن	عليهما	عليهم	عليها
عندما	عندكما	عندكم	عندك	عندئذ	عند	عند
عنكما	عنكم	عنك	عندهن	عندهما	عندهم	عندها
عنو	عنهن	عنهما	عنهم	عنها	عنه	عنم
غيرنا	غيركن	غيركما	غيركم	غيرك	غير	عني
فإحدى	غيري	غيرهن	غيرهما	غيرهم	غيرها	غيره
فإليها	فإليه	فإلينا	فإليكن	فإليكما	فإليك	فإلى
فإنا	فأنا	فإن	فأن	فإليهن	فإليهما	فإليهم
فأنكم	فإنك	فأنك	فانتن	فأنتما	فأنتم	فأنت
فإنه	فأنه	فإننا	فاننا	فإنكما	فأنكما	فإنكم
فأولئك	فإني	فأني	فإنهم	فأنهم	فإنها	فأنها
فاينه	فأيننا	فاينكن	فأينكما	فاينكم	فأينك	فاين
فاذا	فاته	فايني	فأينهن	فأينهما	فأينهم	فاينها
فاللاتي	فالقول	فالغير	فالذين	فالذي	فالتي	فاذن
فان	فالمدة	فاللواتي	فالذين	فاللذان	فاللتين	فاللتان
فبإمكانهن	فبإمكانهم	فبإمكانه	فبإمكاننا	فبإمكانكم	فبإمكان	فانه
فبكما	فبكم	فبكل	فبك	فبالنسبة	فبالإضافة	فبإمكاني
فبهن	فبهما	فبهم	فبها	فبه	فبنسبة	فبنا
فبينما	فبينكن	فبينكما	فبينكم	فبينك	فبين	فبي
فبيني	فبينهن	فبينهما	فبينهم	فبينها	فبينه	فبيننا
فتكون	فتقولون	فتقولوا	فتقولان	فتقول	فتقل	فتحت
فثما	فتينك	فتيك	فتلك	فتكونون	فتكونوا	فتكونان

Continued on next page

Table C.1 – continued from previous page

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
فحينا	فحينئذ	فحين	فحيثما	فحيث	فحاشا	فجاة
فذا	فدائما	فخلال	فحينها	فحينه	فحينما	فحينذاك
فسوف	فسواءا	فسواء	فذينك	فذي	فذلك	فذاك
فضده	فضدنا	فضدكن	فضدكما	فضدكم	فضدك	فضد
فطالما	فضدين	فضدي	فضدهنا	فضدهما	فضدهم	فضدها
فعليكما	فعليكم	فعليك	فعلى	فعدم	فعدة	فعدا
فعليهن	فعليهما	فعليهم	فعليها	فعليه	فعلينا	فعليكن
فعندكما	فعندكم	فعندك	فعندئذ	فعند	فعنا	فعن
فعنكم	فعنك	فعندهن	فعندهما	فعندهم	فعندها	فعندما
فعني	فعنهن	فعنهما	فعنهم	فعنها	فعنه	فعنكما
فغيره	فغيرنا	فغيركن	فغيركما	فغيركم	فغيرك	فغير
ففوقك	ففوق	ففيري	ففيرهن	ففيرهما	ففيرهم	ففيرها
ففوقهم	ففوقها	ففوقه	ففوقنا	ففوقكن	ففوقكما	ففوقكم
ففيكن	ففيكن	ففيكم	ففيك	ففي	ففي	ففوقهما
ففيهن	ففيهما	ففيهم	ففيها	ففيه	ففينا	ففيما
فقط	فقديما	فقد	فقالوا	فقالت	فقالا	فقال
فقولنا	فقولكما	فقولكم	فقولكا	فقولا	فقول	فقلت
فكأننا	فكأنكم	فكأن	فقيلا	فقولهما	فقولها	فقوله
فكأني	فكأنهن	فكأنهما	فكأنهم	فكأنها	فكأنه	فكأننا
فكلا	فكل	فكذلك	فكانوا	فكانتا	فكانت	فكانا
فكلهم	فكلها	فكله	فكلنا	فكلكم	فكلتا	فكلانا
فكيفا	فكيف	فكي	فكمن	فكليهما	فكلينا	فكلهن
فلديه	فلدينا	فلديكم	فلدي	فلدى	فلا	فكيلا
فلست	فلذلك	فلذا	فلديهن	فلديهما	فلديهم	فلديها
فلكل	فلك	فلعل	فلسن	فلستن	فلستما	فلستم
فلم	فلكي	فلكن	فلكما	فلكم	فلكلتا	فلكلا
فلهم	فلهذا	فلها	فله	فلنا	فلن	فلما
فليست	فليس	فلي	فلولا	فلو	فلهن	فلهما
فمدة	فمثلا	فمثل	فمتى	فماذا	فما	فليسوا
فمعنا	فمعكن	فمعكما	فمعكم	فمعك	فمعا	فمع
فمكانكم	فمكانك	فمكان	فمعي	فمعهن	فمعها	فمعه
فمكانهما	فمكانهم	فمكانها	فمكانها	فمكاننا	فمكانكن	فمكانكما
فمن	فمن	فممكنا	فممكن	فمما	فمكاني	فمكانهن
فمنها	فمنه	فمنكن	فمنكم	فمنك	فمنذ	فمنا
فهؤلاء	فنحن	فنتيجة	فمني	فمنهن	فمنهما	فمنهم
فهذه	فهذان	فهذا	فهتين	فهتان	فهاتين	فهاتان
فهناك	فهنا	فهن	فهما	فهم	فهل	فهذين
فوقكن	فوقكما	فوقكم	فوقك	فوق	فهيا	فهي
فيبدو	في	فوقهما	فوقهم	فوقها	فوقه	فوقنا
فيك	فيقولون	فيقولوا	فيقولا	فيقول	فيقل	فيجب
فيمكنك	فيمكن	فيما	فيكونون	فيكونان	فيكون	فيكم
فيمكنها	فيمكنه	فيمكنني	فيمكننا	فيمكنكن	فيمكنكما	فيمكنكم
فيهم	فيها	فيه	فينا	فيمكنهن	فيمكنهما	فيمكنهم
قالوا	قالت	قالا	قال	فيومئذ	فيهن	فيهما
قولا	قول	قلت	قريبا	قديما	قد	قبل
قولهما	قولها	قوله	قولنا	قولكما	قولكم	قولكا
كانه	كاننا	كانكم	كان	كإحدى	كأحد	قيلا

Continued on next page

Table C.1 – continued from previous page

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
كالتي	كافيا	كأني	كأنهن	كأنهما	كأنهم	كأنها
كاللذان	كاللتين	كاللتان	كاللاتي	كالقول	كالذين	كالذي
كانوا	كانتا	كانت	كانا	كان	كاللواتي	كاللذين
كضدنا	كضدكن	كضدكما	كضدكم	كضدك	كضد	كذلك
كضدين	كضدي	كضدهنا	كضدهما	كضدهم	كضدها	كضده
كقوله	كقولنا	كقولكما	كقولكم	كقولكا	كقولا	كقول
كلكم	كلتا	كلانا	كلا	ككل	كقولهما	كقولها
كليهما	كلينا	كلهن	كلهم	كلها	كله	كلنا
لأخر	لإحدى	لأحد	كيلا	كيف	كمن	كما
لأمامه	لأمامنا	لأمامكن	لأمامكما	لأمامكم	لأمامك	لئلا
لأي	لأولئك	لأواخر	لأني	لأن	لأمامي	لأمامها
لابد	لأيهن	لأيهما	لأيهم	لأيها	لأية	لأي
لبعض	لاي	لان	لامامهن	لامامهم	لالقول	لالتي
لتكونان	لتكون	لتقولون	لتقولوا	لتقولان	لتقول	لتقل
لدي	لدى	لتينك	لتيك	لتلك	لتكونون	لتكونوا
لديهن	لديهما	لديهم	لديها	لديه	لدينا	لديكم
لست	لذينك	لذي	لذي	لذلك	لذاك	لذا
لعام	لسوف	لسن	لستن	لستما	لستما	لستم
لقولكم	لقولكا	لقولا	لقول	لقد	لعل	لعدم
لكلا	لكل	لقولهما	لقولها	لقوله	لقولنا	لقولكما
لكيلا	لكي	لكنك	لكن	لكما	لكم	لكلتا
للتين	للتان	للان	للاخ	للاتي	للأمر	للأمام
لما	للواتي	للمزيد	للفن	للذين	للذين	للذان
لمكانكما	لمكانكم	لمكانك	لمكان	لمدى	لمدة	لماذا
لمكاني	لمكانهن	لمكانهما	لمكانهم	لمكانها	لمكاننا	لمكانكن
لهاتان	لها	لهؤلاء	له	له	لنحو	لنا
لهذين	لهذه	لهذان	لهذا	لهتين	لهتان	لهاتين
ليسوا	ليست	ليس	لولا	لهن	لهما	لهم
ليكونان	ليكون	ليقولون	ليقولوا	ليقولا	ليقول	ليقل
ليمكننا	ليمكنكن	ليمكنكما	ليمكنكم	ليمكنك	ليمكن	ليكونون
مئات	ليمكنهن	ليمكنهما	ليمكنهم	ليمكنها	ليمكنه	ليمكنني
مثل	متى	ماهي	ماهو	ماهم	ماذا	مؤكد
معا	مع	مطلقا	مدى	مدة	مثلما	مثلا
معها	معه	معنا	معكن	معكما	معكم	معك
ممكن	مما	ملخ	معين	معي	معهن	معهما
منكم	منك	منذ	منا	من	ممو	ممكنا
مني	منهن	منهما	منهم	منها	منه	منكن
هذا	هاهنا	هاتان	هؤلاء	نحو	نحن	مهما
هنا	هنا	هم	هلا	هذين	هذه	هذه
وان	وإذا	وإذا	هينا	هي	هو	هناك
وبعض	وانما	والى	والتي	واذن	وابن	وان
وحدكن	وحدكما	وحدكم	وحدك	وحتى	وتلك	وتبين
وذلك	وحدي	وحدهن	وحدهما	وحدهم	وحده	وحدنا
وقد	وقد	وقتئذ	وفي	وعلى	وعدم	وربما
ومن	ومدى	ومتى	ولذلك	ولذا	ولا	وكذلك
وهيا	وهي	وهو	وهما	وهم	وهذه	وهذا
يقولا	يقول	يقل	يجب	يبدو	ويني	وهيو

Continued on next page

Table C.1 – continued from previous page

Stopword	Stopword	Stopword	Stopword	Stopword	Stopword	Stopword
يمكنكم	يمكنك	يكونون	يكونان	يكون	يقولون	يقولوا
يمكنهما	يمكنهم	يمكنها	يمكنني	يمكننا	يمكنكن	يمكنكما
					يومئذ	يمكنهن

Appendix C

Appendix D

The Complete Lists of the Score Matrices

	1	2	3	4
[ا](A)	1	1	1	1
[ي](y)	1	1	1	1
[و](w)	1	1	1	1
[ت](t)	1	0	0	1
[ن](n)	1	0	0	1
[م](m)	1	0	0	0
[ل](l)	1	0	0	0
[ه](h)	0	0	0	1
[ة](t)	0	0	0	1
[إ](Ą)	0	0	0	0
[س](s)	0	0	0	0
[أ](Ā)	0	0	0	0
[ى](A)	0	0	0	1
[ئ](Ã)	0	0	0	0
[ء](')	0	0	0	0
[ؤ](w̄)	0	0	0	0
[آ](A)	0	0	0	0
[ف](f)	1	0	0	0
[ب](b)	1	0	0	0
[ك](k)	1	0	0	1

Figure D.1: Score Matrix SM_4 For $patt_4$

	1	2	3	4	5
[ا](A)	1	1	1	1	1
[ي](y)	1	1	1	1	1
[و](w)	1	1	1	1	1
[ت](t)	1	1	1	1	1
[ن](n)	1	1	0	1	1
[م](m)	1	1	0	0	1
[ل](l)	1	1	0	0	0
[ه](h)	0	0	0	1	1
[ة](t)	0	0	0	0	1
[إ](Ą)	0	0	0	0	0
[س](s)	1	0	0	0	0
[أ](Ā)	0	0	0	0	0
[ى](A)	0	0	0	0	1
[ئ](Ã)	0	0	0	1	0
[ء](')	0	0	0	0	1
[ؤ](w̄)	0	0	0	0	0
[آ](A)	0	0	0	0	0
[ف](f)	1	0	0	0	0
[ب](b)	1	1	0	0	0
[ك](k)	1	0	0	1	1

Figure D.2: Score Matrix SM_5 For $patt_5$

	1	2	3	4	5	6
[ا](A)	1	1	1	1	1	1
[ي](y)	1	1	1	1	1	1
[و](w)	1	1	1	1	1	0
[ت](t)	1	1	1	1	1	1
[ن](n)	1	1	1	1	1	1
[م](m)	1	1	1	0	1	1
[ل](l)	1	1	1	0	0	0
[ه](h)	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	1
[إ](Ạ)	0	0	0	0	0	0
[س](s)	1	1	0	0	0	0
[أ](Ā)	0	0	0	0	0	0
[ى](A)	0	0	0	0	0	1
[ئ](Ã)	0	0	1	0	0	0
[ء](')	0	0	0	0	0	1
[ؤ](w̄)	0	0	0	0	0	0
[آ](A)	0	0	0	0	0	0
[ف](f)	1	1	0	0	0	0
[ب](b)	1	0	0	0	0	0
[ك](k)	1	0	0	1	1	1

Figure D.3: Score Matrix SM_6 For $patt_6$

	1	2	3	4	5	6	7
[ا](A)	1	1	1	1	1	1	1
[ي](y)	1	1	1	1	1	1	1
[و](w)	1	1	1	1	1	1	0
[ت](t)	1	1	1	1	1	1	1
[ن](n)	1	1	1	1	1	1	1
[م](m)	1	1	1	1	1	1	1
[ل](l)	1	1	1	0	0	0	0
[ه](h)	0	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	0	1
[إ](Ạ)	0	0	0	0	0	0	0
[س](s)	1	1	1	0	0	0	0
[أ](Ā)	0	0	0	0	0	0	0
[ى](A)	0	0	0	0	0	0	0
[ئ](Ã)	0	0	0	0	1	1	0
[ء](')	0	0	0	0	0	0	1
[ؤ](w̄)	0	0	0	0	0	0	0
[آ](A)	0	0	0	0	0	0	0
[ف](f)	1	0	0	0	0	0	0
[ب](b)	1	0	0	0	0	0	0
[ك](k)	1	0	0	0	1	1	1

Figure D.4: Score Matrix SM_7 For $patt_7$

	1	2	3	4	5	6	7	8
[ا](A)	1	1	1	1	1	1	1	1
[ي](y)	1	1	1	1	1	1	1	1
[و](w)	1	0	0	1	1	1	1	0
[ت](t)	1	1	1	1	1	1	1	1
[ن](n)	0	1	1	1	1	1	1	1
[م](m)	1	1	1	1	1	0	1	1
[ل](l)	1	1	1	0	0	0	0	0
[ه](h)	0	0	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	0	0	1
[إ](Ą)	0	0	0	0	0	0	0	0
[س](s)	0	1	1	0	0	0	0	0
[أ](Ā)	0	0	0	0	0	0	0	0
[ى](A)	0	0	0	0	0	0	0	0
[ئ](Ā)	0	0	0	0	0	1	1	0
[ء](')	0	0	0	0	0	0	0	1
[ؤ](w̄)	0	0	0	0	0	0	0	0
[آ](A)	0	0	0	0	0	0	0	0
[ف](f)	1	0	0	0	0	0	0	0
[ب](b)	1	0	0	0	0	0	0	0
[ك](k)	1	0	0	0	0	1	0	1

Figure D.5: Score Matrix SM_8 For $patt_8$

	1	2	3	4	5	6	7	8	9
[ا](A)	1	1	1	1	1	1	1	1	1
[ي](y)	1	1	1	1	1	1	1	1	1
[و](w)	1	0	1	1	1	1	1	1	0
[ت](t)	1	1	1	1	1	1	1	1	1
[ن](n)	0	1	1	1	1	1	1	1	1
[م](m)	1	1	1	1	1	1	0	1	1
[ل](l)	1	1	1	1	0	0	0	1	1
[ه](h)	0	0	0	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	0	0	0	1
[إ](Ą)	1	1	1	1	1	0	0	0	0
[س](s)	1	1	1	1	1	0	0	0	0
[أ](Ā)	1	1	1	1	1	0	0	0	0
[ى](A)	0	0	0	0	0	0	0	0	1
[ئ](Ā)	0	0	0	1	0	1	1	1	0
[ء](')	0	0	0	0	0	0	1	0	1
[ؤ](w̄)	0	0	0	0	0	0	0	0	0
[آ](A)	0	0	0	0	0	0	0	0	0
[ف](f)	1	1	0	0	0	0	0	0	0
[ب](b)	1	1	0	0	0	0	0	0	0
[ك](k)	1	1	0	0	0	0	0	1	1

Figure D.6: Score Matrix SM_9 For $patt_9$

	1	2	3	4	5	6	7	8	9	10
[ا](A)	1	1	1	1	1	1	1	1	1	1
[ي](y)	1	1	1	1	1	1	1	1	1	1
[و](w)	1	0	0	1	1	1	1	1	1	0
[ت](t)	1	1	1	1	1	1	1	1	1	1
[ن](n)	0	1	1	1	1	1	1	1	1	1
[م](m)	1	1	1	1	1	1	0	0	1	1
[ل](l)	1	1	1	1	0	0	0	0	0	0
[ه](h)	0	0	0	0	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	0	0	0	0	1
[إ](Ā)	0	0	0	0	0	0	0	0	0	0
[س](s)	1	1	1	1	1	1	0	0	0	0
[أ](Ā)	0	0	0	0	0	0	0	0	0	0
[ى](Ā)	0	0	0	0	0	0	0	0	0	1
[ئ](Ā)	0	0	0	0	0	0	0	0	0	0
[ء](')	0	0	0	0	0	0	0	0	0	0
[ؤ](w̄)	0	0	0	0	0	0	0	0	0	0
[آ](Ā)	0	0	0	0	0	0	0	0	0	0
[ف](f)	1	0	0	0	0	0	0	0	0	0
[ب](b)	1	1	0	0	0	0	0	0	0	0
[ك](k)	1	0	0	0	0	0	0	0	1	0

Figure D.7: Score Matrix SM_{10} For $patt_{10}$

	1	2	3	4	5	6	7	8	9	10	11
[ا](A)	1	1	1	1	1	1	1	1	1	1	1
[ي](y)	0	1	0	0	0	0	1	1	1	1	1
[و](w)	1	0	0	0	1	1	1	1	0	1	0
[ت](t)	0	1	1	1	1	1	1	1	1	1	1
[ن](n)	0	1	1	1	1	0	0	1	1	1	1
[م](m)	0	1	1	1	1	0	1	0	0	1	1
[ل](l)	1	1	1	1	0	0	0	0	0	0	0
[ه](h)	0	0	0	0	0	0	0	0	1	1	1
[ة](t)	0	0	0	0	0	0	0	0	0	0	1
[إ](Ā)	0	0	0	0	0	0	0	0	0	0	0
[س](s)	1	1	1	1	1	1	0	0	0	0	0
[أ](Ā)	0	0	0	0	0	0	0	0	0	0	0
[ى](A)	0	0	0	0	0	0	0	0	0	0	0
[ئ](Ā)	0	0	0	0	0	0	0	0	0	0	0
[ء](')	0	0	0	0	0	0	0	0	0	0	0
[ؤ](w̄)	0	0	0	0	0	0	0	0	0	0	0
[آ](A)	0	0	0	0	0	0	0	0	0	0	0
[ف](f)	1	0	0	0	0	0	0	0	0	0	0
[ب](b)	1	1	0	0	0	0	0	0	0	0	0
[ك](k)	1	0	0	0	0	0	0	0	0	1	0

Figure D.8: Score Matrix SM_{11} For $patt_{11}$

	1	2	3	4	5	6	7	8	9	10	11	12
[ا](A)	1	1	1	1	1	1	1	1	1	1	1	1
[ي](y)	0	0	0	0	0	0	0	1	0	1	1	0
[و](w)	1	0	0	0	0	0	1	1	1	1	1	0
[ت](t)	0	0	0	1	1	1	1	1	0	1	0	1
[ن](n)	0	0	0	0	0	0	0	0	1	0	1	1
[م](m)	0	0	0	1	1	1	0	0	1	0	0	1
[ل](l)	1	1	1	1	0	0	0	0	0	0	0	0
[ه](h)	0	0	0	0	0	0	0	0	0	0	1	0
[ة](t)	0	0	0	0	0	0	0	0	0	0	0	1
[إ](Ạ)	0	0	0	0	0	0	0	0	0	0	0	0
[س](s)	0	0	1	1	1	1	0	0	0	0	0	0
[آ](Ā)	0	0	0	0	0	0	0	0	0	0	0	0
[ى](A)	0	0	0	0	0	0	0	0	0	0	0	0
[ئ](Ā)	0	0	0	0	0	0	0	0	0	0	0	0
[ء](')	0	0	0	0	0	0	0	0	0	0	0	0
[ؤ](w̄)	0	0	0	0	0	0	0	0	0	0	0	0
[أ](A)	0	0	0	0	0	0	0	0	0	0	0	0
[ف](f)	1	0	0	0	0	0	0	0	0	0	0	0
[ب](b)	1	1	0	0	0	0	0	0	0	0	0	0
[ك](k)	1	0	0	0	0	0	0	0	0	0	1	0

Figure D.9: Score Matrix SM_{12} For $patt_{12}$

Appendix E

The Complete List of the Target Encoding

Table E.1: The Complete Target Encoded for $BPNN_4$

Token's Length	Root Letter Positions	Target Vector	Class
4	1, 2, 3	0001	1
4	1, 2, 4	0010	2
4	1, 3, 4	0100	3
4	2, 3, 4	1000	4

Table E.2: The Complete Target Encoded for $BPNN_5$

Token's Length	Root Letter Positions	Target Vector	Class
5	1, 2, 3	0000000001	1
5	1, 2, 4	0000000010	2
5	1, 2, 5	0000000100	3
5	1, 3, 4	0000001000	4
5	1, 3, 5	0000010000	5
5	1, 4, 5	0000100000	6
5	2, 3, 4	0001000000	7
5	2, 3, 5	0010000000	8
5	2, 4, 5	0100000000	9
5	3, 4, 5	1000000000	10

Table E.3: The Complete Target Encoded for $BPNN_6$

Token's Length	Root Letter Positions	Target Vector	Class
6	1, 2, 3	00000000000001	1
6	1, 2, 4	00000000000010	2
6	1, 3, 4	00000000000100	3
6	1, 4, 6	00000000001000	4
6	2, 3, 4	00000000010000	5
6	2, 3, 5	00000000100000	6
6	2, 3, 6	00000001000000	7
6	2, 4, 5	00000010000000	8
6	2, 4, 6	00000100000000	9
6	2, 5, 6	00001000000000	10
6	3, 4, 5	00010000000000	11
6	3, 4, 6	00100000000000	12
6	3, 5, 6	01000000000000	13
6	4, 5, 6	10000000000000	14

Table E.4: The Complete Target Encoded for $BPNN_7$

Token's Length	Root Letter Positions	Target Vector	Class
7	1, 2, 3	0000000000000001	1
7	1, 2, 4	0000000000000010	2
7	1, 3, 4	0000000000000100	3
7	2, 3, 4	0000000000001000	4
7	2, 3, 5	0000000000010000	5
7	2, 4, 5	0000000000100000	6
7	2, 4, 6	0000000001000000	7
7	3, 4, 5	0000000010000000	8
7	3, 4, 6	0000000100000000	9
7	3, 4, 7	0000001000000000	10
7	3, 5, 6	0000010000000000	11
7	3, 5, 7	0000100000000000	12
7	4, 5, 6	0001000000000000	13
7	4, 5, 7	0010000000000000	14
7	4, 6, 7	0100000000000000	15
7	5, 6, 7	1000000000000000	16

Table E.5: The Complete Target Encoded for $BPNN_8$

Token's Length	Root Letter Positions	Target Vector	Class
8	1, 2, 3	00000000000000000001	1
8	1, 2, 4	00000000000000000010	2
8	2, 3, 4	00000000000000000100	3
8	2, 3, 5	00000000000000001000	4
8	2, 4, 5	00000000000000010000	5

table continued on next page

Table E.5 – *continued from previous page*

Token's Length	Root Letter Positions	Target Vector	Class
8	2, 4, 6	0000000000000100000	6
8	3, 4, 5	0000000000001000000	7
8	3, 4, 6	0000000000010000000	8
8	3, 5, 6	0000000000100000000	9
8	3, 5, 7	0000000001000000000	10
8	3, 6, 8	0000000010000000000	11
8	4, 5, 6	0000000100000000000	12
8	4, 5, 7	0000001000000000000	13
8	4, 5, 8	0000010000000000000	14
8	4, 6, 7	0000100000000000000	15
8	4, 6, 8	0001000000000000000	16
8	5, 6, 7	0010000000000000000	17
8	5, 6, 8	0100000000000000000	18
8	5, 7, 8	1000000000000000000	19

Table E.6: The Complete Target Encoded for $BPNN_9$

Token's Length	Root Letter Positions	Target Vector	Class
9	1, 2, 3	000000000000000000000000000001	1
9	1, 2, 4	000000000000000000000000000010	2
9	1, 2, 5	000000000000000000000000000100	3
9	1, 5, 7	000000000000000000000000001000	4
9	2, 3, 4	000000000000000000000000010000	5
9	2, 3, 5	000000000000000000000000100000	6
9	2, 4, 5	000000000000000000000001000000	7
9	2, 4, 6	000000000000000000000010000000	8
9	2, 5, 7	000000000000000000000100000000	9
9	3, 4, 5	000000000000000000001000000000	10
9	3, 4, 6	000000000000000000010000000000	11
9	3, 4, 7	000000000000000000100000000000	12
9	3, 5, 6	000000000000000001000000000000	13
9	3, 5, 7	000000000000000010000000000000	14
9	4, 5, 6	000000000000000100000000000000	15
9	4, 5, 7	000000000000001000000000000000	16
9	4, 5, 8	000000000000010000000000000000	17
9	4, 6, 7	000000000000100000000000000000	18
9	4, 6, 8	000000000001000000000000000000	19
9	4, 7, 8	000000000010000000000000000000	20
9	4, 7, 9	000000000100000000000000000000	21
9	5, 6, 7	000000001000000000000000000000	22
9	5, 6, 8	000000010000000000000000000000	23
9	5, 6, 9	000000100000000000000000000000	24
9	5, 7, 8	000001000000000000000000000000	25
9	5, 7, 9	000010000000000000000000000000	26
9	6, 7, 8	000100000000000000000000000000	27

table continued on next page

Table E.6 – *continued from previous page*

Token's Length	Root Letter Positions	Target Vector	Class
9	6, 7, 9	00100000000000000000000000000	28
9	6, 8, 9	01000000000000000000000000000	29
9	7, 8, 9	10000000000000000000000000000	30

Table E.7: The Complete Target Encoded for $BPNN_{10}$

Token's Length	Root Letter Positions	Target Vector	Class
10	1, 7, 8	0000000000000000000000001	1
10	2, 3, 4	0000000000000000000000010	2
10	2, 3, 5	0000000000000000000000100	3
10	2, 4, 5	0000000000000000000001000	4
10	2, 4, 6	0000000000000000000010000	5
10	3, 4, 5	0000000000000000000100000	6
10	3, 4, 6	0000000000000000001000000	7
10	3, 5, 6	0000000000000000010000000	8
10	3, 5, 7	0000000000000000100000000	9
10	3, 6, 8	0000000000000001000000000	10
10	4, 5, 6	0000000000000010000000000	11
10	4, 5, 7	0000000000000100000000000	12
10	4, 5, 8	0000000000001000000000000	13
10	4, 6, 7	0000000000010000000000000	14
10	4, 6, 8	0000000000100000000000000	15
10	4, 7, 8	0000000001000000000000000	16
10	5, 6, 7	0000000010000000000000000	17
10	5, 6, 8	0000000100000000000000000	18
10	5, 7, 8	0000001000000000000000000	19
10	5, 7, 9	0000010000000000000000000	20
10	6, 7, 8	0000100000000000000000000	21
10	6, 7, 9	0001000000000000000000000	22
10	6, 8, 9	0010000000000000000000000	23
10	6, 8, 10	0100000000000000000000000	24
10	7, 8, 9	1000000000000000000000000	25
10	7, 8, 10	1000000000000000000000000	26

Table E.8: The Complete Target Encoded for $BPNN_{11}$

Token's Length	Root Letter Positions	Target Vector	Class
11	2, 4, 6	00000000000000001	1
11	3, 4, 6	00000000000000010	2
11	3, 5, 6	00000000000000100	3
11	3, 5, 7	00000000000001000	4
11	4, 5, 6	00000000000010000	5
11	4, 5, 7	00000000000100000	6

table continued on next page

Table E.8 – *continued from previous page*

Token's Length	Root Letter Positions	Target Vector	Class
11	4, 6, 8	0000000001000000	7
11	5, 6, 7	0000000010000000	8
11	5, 6, 8	0000000100000000	9
11	5, 6, 9	0000001000000000	10
11	5, 7, 8	0000010000000000	11
11	5, 7, 9	0000100000000000	12
11	6, 7, 8	0001000000000000	13
11	6, 7, 9	0010000000000000	14
11	6, 8, 9	0100000000000000	15
11	7, 8, 9	1000000000000000	16
11	8, 9, 10	1000000000000000	17

Appendix F

Number of Neurons in the Hidden Layers

Table F.1: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_4$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
4	0.1145561	0.1148164	0.1216486	0.1143701	0.1193599	0.1169502
5	0.1113855	0.1247833	0.1115374	0.1137931	0.1113290	0.1145657
6	0.1030022	0.1113334	0.1090625	0.1101775	0.1127836	0.1092718
7	0.1067190	0.1091654	0.1066213	0.1048473	0.1039767	0.1062659
8	0.1039348	0.0975601	0.0945494	0.1044778	0.1063969	0.1013838
9	0.0996528	0.0937576	0.1016259	0.1049016	0.1037443	0.1007364
10	0.0959925	0.0977803	0.0936786	0.0933737	0.0942179	0.0950086
11	0.0953250	0.1118523	0.0907649	0.0948221	0.0951151	0.0975759
12	0.0926278	0.0909209	0.0902183	0.0924597	0.0947102	0.0921874
13	0.0899816	0.0904081	0.0924370	0.0879367	0.0934493	0.0908425
14	0.0900568	0.0908192	0.0941017	0.0946975	0.1017401	0.0942831
15	0.0961057	0.0915545	0.0887634	0.0909342	0.0894662	0.0913648
16	0.0914085	0.0879125	0.0909122	0.0899075	0.0893352	0.0898952
17	0.0877792	0.0916783	0.0922123	0.0970942	0.0882738	0.0914076
18	0.0877161	0.0909515	0.0903733	0.0991982	0.0858846	0.0908247
19	0.0989957	0.0917288	0.0903173	0.0882167	0.0877338	0.0913985
20	0.0888636	0.0893121	0.0918632	0.0868360	0.0925849	0.0898920
21	0.0907992	0.0943698	0.0919475	0.0892845	0.0932196	0.0919241
22	0.0877585	0.0899058	0.0903428	0.0912020	0.0857219	0.0889862
23	0.0890485	0.0897936	0.0876362	0.0909045	0.0879124	0.0890590
24	0.0877001	0.0881523	0.0842669	0.0848657	0.0867751	0.0863520
25	0.0872173	0.0841889	0.0889574	0.0876747	0.0876458	0.0871368
26	0.0957700	0.0883402	0.0883138	0.0896394	0.0880221	0.0900171
27	0.0878897	0.0866275	0.0870780	0.0860570	0.0866096	0.0868524

table continued on next page

Table F.1 – *continued from previous page*

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
28	0.0870180	0.0872560	0.0884963	0.0880741	0.0892668	0.0880222
29	0.0865603	0.0888585	0.0891339	0.0818730	0.0856603	0.0864172
30	0.0836018	0.0860587	0.0877835	0.0872850	0.0867758	0.0863010
31	0.0912445	0.0854871	0.0814105	0.0858626	0.0822439	0.0852497
32	0.0875797	0.0873709	0.0876831	0.0848809	0.0878095	0.0870648
33	0.0864475	0.0902359	0.0834274	0.0862727	0.0851515	0.0863070
34	0.0887363	0.0893111	0.0858644	0.0966402	0.0851859	0.0891476
35	0.0874258	0.0830726	0.0900899	0.0874742	0.0870544	0.0870234
36	0.0916865	0.0862568	0.0942285	0.0842586	0.0863117	0.0885484
37	0.0878176	0.0828580	0.0864355	0.0847201	0.0861602	0.0855983
38	0.0855029	0.0846637	0.0848990	0.0840298	0.0867500	0.0851691
39	0.0853720	0.0868711	0.0903840	0.0836312	0.0855300	0.0863577
40	0.0835249	0.0814149	0.0863844	0.0830406	0.0832820	0.0835294
41	0.0912577	0.0887260	0.0846866	0.0856289	0.0871020	0.0874802
42	0.0873604	0.0857965	0.0849970	0.0839161	0.0849319	0.0854004
43	0.0871074	0.0884211	0.0856455	0.0897413	0.0885783	0.0878987

Table F.2: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_5$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
7	0.0443230	0.0437787	0.0429466	0.0429478	0.0476984	0.0443389
8	0.0414928	0.0456753	0.0432538	0.0424768	0.0418751	0.0429548
9	0.0409235	0.0426993	0.0415145	0.0415395	0.0419044	0.0417162
10	0.0402235	0.0400012	0.0433321	0.0413066	0.0414931	0.0412713
11	*0.0393634*	*0.0398022*	0.0400630	0.0387370	0.0431608	0.0402253
12	0.0386917	0.0385418	0.0390267	0.0372767	*0.0393744*	*0.0385823*
13	0.0389898	0.0382088	0.0368707	0.0430265	0.0427704	0.0399732
14	0.0374015	0.0363060	0.0379915	0.0373205	0.0365891	0.0371217
15	0.0377461	0.0369306	0.0356944	0.0420304	0.0398077	0.0384418
16	0.0353791	0.0381512	0.0369974	0.0370043	0.0360825	0.0367229
17	0.0376531	0.0372925	0.0344551	0.0370557	0.0355964	0.0364106
18	0.0361177	0.0399140	0.0358893	0.0352971	0.0350297	0.0364496
19	0.0356802	0.0342741	0.0359934	0.0338314	0.0374617	0.0354482
20	0.0358318	0.0357735	0.0344687	0.0376144	0.0355809	0.0358539
21	0.0346524	0.0349558	0.0343407	0.0346175	0.0362280	0.0349589
22	0.0354935	0.0355986	0.0348117	0.0347965	0.0352426	0.0351886
23	0.0327927	0.0351951	0.0347185	0.0360413	0.0350626	0.0347620
24	0.0351463	0.0339845	0.0334551	0.0346053	0.0306731	0.0335729
25	0.0346484	0.0343498	0.0323986	0.0336225	0.0350150	0.0340069
26	0.0323327	0.0328859	0.0330716	0.0344407	0.0331479	0.0331758
27	0.0332415	0.0319063	0.0329427	0.0329529	0.0336600	0.0329407
28	0.0338056	0.0321367	0.0326851	0.0344827	0.0311676	0.0328555
29	0.0340544	0.0340433	0.0332957	0.0338113	0.0342078	0.0338825
30	0.0349507	0.0335202	0.0337488	0.0312457	0.0372805	0.0341492
31	0.0335892	0.0308710	0.0345456	0.0369231	0.0334312	0.0338720
32	0.0324069	0.0333543	0.0321398	0.0331591	0.0325232	0.0327167
33	0.0303067	0.0322422	0.0329227	0.0322007	0.0315230	0.0318391
34	0.0338801	0.0327690	0.0296263	0.0325016	0.0316748	0.0320904
35	0.0321734	0.0307026	0.0311874	0.0326272	0.0336438	0.0320669
36	0.0316122	0.0315880	0.0319749	0.0351578	0.0329166	0.0326499
37	0.0305314	0.0332158	0.0313328	0.0311277	0.0327528	0.0317921
38	0.0319560	0.0338520	0.0311653	0.0319372	0.0307763	0.0319374
39	0.0320761	0.0310056	0.0306196	0.0302796	0.0322331	0.0312428
40	0.0313338	0.0319689	0.0306834	0.0324834	0.0311541	0.0315247
41	0.0320363	0.0312327	0.0311327	0.0319039	0.0345838	0.0321779
42	0.0299444	0.0308592	0.0313784	0.0310629	0.0317730	0.0310036
43	0.0307254	0.0307119	0.0316174	0.0314766	0.0317164	0.0312495
44	0.0308951	0.0310249	0.0293420	0.0315660	0.0312588	0.0308174
45	0.0306230	0.0318366	0.0327690	0.0310317	0.0313473	0.0315215
46	0.0318264	0.0303799	0.0322833	0.0300659	0.0315228	0.0312157

Table F.3: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_6$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
9	0.0224424	0.0210838	0.0229774	0.0226279	0.0223078	0.0222879
10	0.0195467	0.0207228	0.0232798	0.0198659	0.0211429	0.0209116
11	0.0177830	0.0197726	0.0190892	0.0191474	0.0221086	0.0195802
12	0.0190489	0.0204728	0.0202197	0.0179205	0.0181829	0.0191690
13	0.0185515	0.0177541	0.0207730	0.0172378	0.0176209	0.0183875
14	0.0182836	0.0186373	0.0178434	0.0181468	0.0163991	0.0178620
15	0.0194926	0.0166354	0.0170268	0.0157731	0.0169434	0.0171743
16	0.0173659	0.0166879	0.0164060	0.0176656	0.0161312	0.0168513
17	0.0174283	0.0149013	0.0159214	0.0152208	0.0164679	0.0159879
18	0.0159053	0.0156807	0.0148829	0.0174979	0.0160711	0.0160076
19	0.0145469	0.0149062	0.0163242	0.0157165	0.0167490	0.0156486
20	0.0146649	0.0154205	0.0143300	0.0148997	0.0151452	0.0148921
21	0.0154710	0.0158112	0.0152686	0.0146935	0.0141777	0.0150844
22	0.0151307	0.0161826	0.0142900	0.0140908	0.0135404	0.0146469
23	0.0140415	0.0141506	0.0142791	0.0137121	0.0141429	0.0140652
24	0.0135940	0.0146003	0.0144418	0.0135797	0.0146887	0.0141809
25	0.0141658	0.0142596	0.0152405	0.0137100	0.0140321	0.0142816
26	0.0123874	0.0135677	0.0127461	0.0143610	0.0130589	0.0132242
27	0.0137516	0.0122709	0.0134249	0.0133943	0.0153726	0.0136429
28	0.0143700	0.0135922	0.0134447	0.0135481	0.0135062	0.0136922
29	0.0125105	0.0139468	0.0130543	0.0136155	0.0145946	0.0135443
30	0.0140627	0.0128329	0.0138143	0.0127690	0.0131633	0.0133284
31	0.0128740	0.0119581	0.0135398	0.0125461	0.0130654	0.0127967
32	0.0134344	0.0129222	0.0118159	0.0134216	0.0136398	0.0130468
33	0.0125871	0.0126810	0.0130288	0.0122503	0.0119869	0.0125068
34	0.0120613	0.0125356	0.0123844	0.0123864	0.0127249	0.0124185
35	0.0114160	0.0118159	0.0127211	0.0118165	0.0129177	0.0121374
36	0.0118418	0.0127815	0.0124513	0.0124233	0.0121255	0.0123247
37	0.0124413	0.0129106	0.0126779	0.0121208	0.0118520	0.0124005
38	0.0127337	0.0128371	0.0117308	0.0124474	0.0120290	0.0123556
39	0.0120500	0.0118720	0.0124018	0.0136709	0.0131768	0.0126343
40	0.0117219	0.0125458	0.0113698	0.0124464	0.0116531	0.0119474
41	0.0124052	0.0131734	0.0112343	0.0119523	0.0122712	0.0122073
42	0.0113835	0.0130048	0.0128576	0.0116397	0.0118481	0.0121467
43	0.0117489	0.0116436	0.0115872	0.0113368	0.0114905	0.0115614
44	0.0116526	0.0125114	0.0122263	0.0118002	0.0114819	0.0119345
45	0.0109630	0.0113118	0.0115704	0.0132500	0.0116495	0.0117489
46	0.0125149	0.0115119	0.0117751	0.0119775	0.0105654	0.0116690
47	0.0116160	0.0125461	0.0121925	0.0115848	0.0125691	0.0121017
48	0.0123420	0.0121121	0.0123740	0.0115922	0.0115161	0.0119873

Table F.4: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_7$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
11	0.0144951	0.0141216	0.0147358	0.0167162	0.0153248	0.0150787
12	0.0125835	0.0134589	0.0136814	0.0134157	0.0141637	0.0134606
13	0.0135741	0.0163001	0.0135175	0.0151407	0.0129526	0.0142970
14	0.0122843	0.0130227	0.0124838	0.0118861	0.0124160	0.0124186
15	0.0105580	0.0104393	0.0117788	0.0117258	0.0116032	0.0112210
16	0.0116428	0.0114390	0.0111891	0.0101759	0.0117834	0.0112460
17	0.0116323	0.0107449	0.0115705	0.0110397	0.0110313	0.0112037
18	0.0093623	0.0095103	0.0108707	0.0109372	0.0106182	0.0102597
19	0.0090425	0.0100836	0.0113659	0.0095266	0.0108927	0.0101823
20	0.0097807	0.0093373	0.0085414	0.0091218	0.0087655	0.0091093
21	0.0078559	0.0074859	0.0089318	0.0092459	0.0088994	0.0084838
22	0.0085928	0.0085518	0.0083883	0.0085240	0.0097182	0.0087550
23	0.0084839	0.0078702	0.0073655	0.0088211	0.0080855	0.0081252
24	0.0086781	0.0076473	0.0085097	0.0077695	0.0079712	0.0081152
25	0.0078931	0.0084462	0.0073690	0.0081621	0.0090166	0.0081774
26	0.0082479	0.0072060	0.0080235	0.0074527	0.0078135	0.0077487
27	0.0076918	0.0080475	0.0070746	0.0074448	0.0086071	0.0077732
28	0.0074271	0.0083396	0.0079725	0.0073118	0.0074783	0.0077059
29	0.0071765	0.0065441	0.0081803	0.0076079	0.0070346	0.0073087
30	0.0068782	0.0068704	0.0068481	0.0064183	0.0069752	0.0067980
31	0.0067557	0.0071494	0.0073743	0.0076826	0.0060190	0.0069962
32	0.0068773	0.0070701	0.0061709	0.0062151	0.0071462	0.0066959
33	0.0072156	0.0067428	0.0065627	0.0057771	0.0061310	0.0064858
34	0.0067698	0.0061267	0.0060136	0.0064637	0.0060175	0.0062783
35	0.0067131	0.0061771	0.0069879	0.0066692	0.0065730	0.0066241
36	0.0062995	0.0060850	0.0053995	0.0070765	0.0061156	0.0061952
37	0.0052795	0.0058514	0.0061578	0.0069191	0.0066102	0.0061636
38	0.0069430	0.0058754	0.0054716	0.0062052	0.0059225	0.0060835
39	0.0054980	0.0057899	0.0070112	0.0067685	0.0067776	0.0063690
40	0.0059775	0.0054425	0.0057549	0.0059651	0.0053841	0.0057048
41	0.0059001	0.0058667	0.0058501	0.0057061	0.0058952	0.0058436
42	0.0062076	0.0055512	0.0053302	0.0062715	0.0051790	0.0057079
43	0.0049850	0.0060709	0.0045987	0.0051343	0.0057090	0.0052996
44	0.0054849	0.0063676	0.0056883	0.0056658	0.0057747	0.0057963
45	0.0054804	0.0053856	0.0052589	0.0062581	0.0055023	0.0055771
46	0.0051963	0.0061866	0.0066759	0.0059672	0.0053083	0.0058669
47	0.0053653	0.0054814	0.0053788	0.0055927	0.0059099	0.0055456
48	0.0052866	0.0062992	0.0054184	0.0060338	0.0055293	0.0057135
49	0.0050847	0.0050436	0.0053885	0.0048633	0.0049988	0.0050758
50	0.0051810	0.0058704	0.0050439	0.0046798	0.0051361	0.0051822

Table F.5: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_8$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
12	0.0127607	0.0116894	0.0111754	0.0115826	0.0119845	0.0118385
13	0.0109995	0.0120175	0.0116914	0.0103148	0.0124733	0.0114993
14	0.0107434	0.0100709	0.0104723	0.0107739	0.0110302	0.0106181
15	0.0097049	0.0095899	0.0096263	0.0101579	0.0102647	0.0098687
16	0.0099476	0.0086213	0.0102090	0.0099290	0.0090144	0.0095443
17	0.0085548	0.0106606	0.0103414	0.0083687	0.0093316	0.0094514
18	0.0092864	0.0079722	0.0088306	0.0077230	0.0079278	0.0083480
19	0.0090093	0.0091048	0.0077068	0.0083150	0.0084307	0.0085133
20	0.0084317	0.0081834	0.0076076	0.0072876	0.0077524	0.0078525
21	0.0080522	0.0082199	0.0077298	0.0081963	0.0087545	0.0081905
22	0.0069703	0.0081204	0.0078260	0.0086197	0.0075480	0.0078169
23	0.0074100	0.0083442	0.0078319	0.0066651	0.0081279	0.0076758
24	0.0073837	0.0069496	0.0062584	0.0068302	0.0070251	0.0068894
25	0.0068882	0.0073992	0.0069105	0.0067784	0.0068516	0.0069656
26	0.0066179	0.0070567	0.0072252	0.0065140	0.0067833	0.0068394
27	0.0065042	0.0064082	0.0067552	0.0071524	0.0068091	0.0067258
28	0.0066259	0.0069627	0.0064544	0.0064970	0.0072116	0.0067503
29	0.0062360	0.0067058	0.0069421	0.0065303	0.0065372	0.0065903
30	0.0063148	0.0058591	0.0058075	0.0058215	0.0061545	0.0059915
31	0.0067699	0.0062245	0.0062816	0.0064564	0.0057427	0.0062950
32	0.0052717	0.0056855	0.0057857	0.0058034	0.0056902	0.0056473
33	0.0060092	0.0059941	0.0068776	0.0058558	0.0055146	0.0060503
34	0.0057699	0.0055179	0.0058322	0.0057813	0.0057549	0.0057312
35	0.0058625	0.0059630	0.0057639	0.0054227	0.0053370	0.0056698
36	0.0050934	0.0058664	0.0057409	0.0055457	0.0053247	0.0055142
37	0.0055097	0.0047524	0.0050203	0.0059528	0.0057143	0.0053899
38	0.0054124	0.0057605	0.0050444	0.0048946	0.0059369	0.0054098
39	0.0057420	0.0056247	0.0051647	0.0053548	0.0051541	0.0054081
40	0.0048694	0.0051318	0.0050329	0.0045989	0.0048336	0.0048933
41	0.0054577	0.0054479	0.0046920	0.0045792	0.0051397	0.0050633
42	0.0053061	0.0053186	0.0051645	0.0049041	0.0051884	0.0051763
43	0.0046546	0.0045578	0.0049413	0.0050645	0.0047271	0.0047891
44	0.0055018	0.0047095	0.0053061	0.0048314	0.0049092	0.0050516
45	0.0045265	0.0051593	0.0057230	0.0047332	0.0050295	0.0050343
46	0.0055104	0.0046071	0.0049843	0.0050646	0.0050864	0.0050506
47	0.0046313	0.0051450	0.0052222	0.0049314	0.0047728	0.0049405
48	0.0048693	0.0046778	0.0043402	0.0041946	0.0045848	0.0045333
49	0.0043709	0.0042869	0.0048762	0.0045997	0.0047327	0.0045733
50	0.0043027	0.0045700	0.0048192	0.0047253	0.0046050	0.0046044
51	0.0047744	0.0044524	0.0046198	0.0050718	0.0041086	0.0046054

Table F.6: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_9$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
16	0.0151856	0.0139826	0.0070508	0.0085967	0.0065716	0.0102775
17	0.0068168	0.0069630	0.0068923	0.0061900	0.0070015	0.0067727
18	0.0065891	0.0064925	0.0062342	0.0063977	0.0067242	0.0064875
19	0.0074024	0.0064189	0.0074376	0.0060662	0.0058962	0.0066443
20	0.0056883	0.0061803	0.0068794	0.0065117	0.0057056	0.0061931
21	0.0062862	0.0063350	0.0060405	0.0059165	0.0056232	0.0060403
22	0.0056424	0.0066780	0.0064414	0.0056077	0.0063228	0.0061385
23	0.0061385	0.0057325	0.0063551	0.0054751	0.0051722	0.0057747
24	0.0051457	0.0057031	0.0050702	0.0055054	0.0054114	0.0053672
25	0.0053625	0.0048067	0.0050900	0.0046790	0.0049652	0.0049807
26	0.0047654	0.0046709	0.0047573	0.0050711	0.0050622	0.0048654
27	0.0052510	0.0051626	0.0052520	0.0042270	0.0052070	0.0050199
28	0.0045910	0.0047505	0.0042221	0.0046994	0.0053150	0.0047156
29	0.0042344	0.0043992	0.0041525	0.0045639	0.0048135	0.0044327
30	0.0042115	0.0047434	0.0039454	0.0043100	0.0046931	0.0043807
31	0.0039148	0.0043722	0.0043817	0.0046498	0.0045224	0.0043682
32	0.0041514	0.0040939	0.0039737	0.0044639	0.0039616	0.0041289
33	0.0037038	0.0037882	0.0042224	0.0042426	0.0041069	0.0040128
34	0.0035943	0.0040459	0.0041131	0.0040407	0.0042693	0.0040127
35	0.0036101	0.0040276	0.0040539	0.0042698	0.0042647	0.0040452
36	0.0037823	0.0035913	0.0032932	0.0038866	0.0039654	0.0037038
37	0.0035611	0.0040697	0.0036462	0.0037626	0.0035329	0.0037145
38	0.0036534	0.0039360	0.0036207	0.0042054	0.0034446	0.0037720
39	0.0032603	0.0032574	0.0034938	0.0035333	0.0036276	0.0034345
40	0.0033597	0.0038707	0.0038987	0.0031708	0.0036259	0.0035852
41	0.0034263	0.0034188	0.0032082	0.0038790	0.0034349	0.0034734
42	0.0032695	0.0030175	0.0038296	0.0032183	0.0032136	0.0033097
43	0.0033036	0.0035452	0.0035775	0.0034387	0.0031624	0.0034055
44	0.0036776	0.0032399	0.0034645	0.0032671	0.0032150	0.0033728
45	0.0030660	0.0031675	0.0033745	0.0031172	0.0027402	0.0030931
46	0.0030640	0.0034759	0.0030405	0.0033074	0.0034140	0.0032604
47	0.0033355	0.0031328	0.0031576	0.0032127	0.0032116	0.0032100
48	0.0026395	0.0031148	0.0029261	0.0032290	0.0032543	0.0030327
49	0.0030113	0.0031396	0.0027308	0.0030586	0.0034069	0.0030694
50	0.0030700	0.0030407	0.0031181	0.0031054	0.0033086	0.0031286
51	0.0028796	0.0032319	0.0027426	0.0032445	0.0028298	0.0029857
52	0.0031398	0.0028337	0.0028495	0.0032580	0.0031094	0.0030381
53	0.0023466	0.0031680	0.0027858	0.0026038	0.0028704	0.0027549
54	0.0031688	0.0029821	0.0029999	0.0028153	0.0025723	0.0029077
55	0.0032302	0.0032727	0.0030249	0.0027957	0.0028891	0.0030425

Table F.7: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_{10}$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
16	0.0015783	0.0011639	0.0015985	0.0009758	0.0013786	0.0013390
17	0.0013624	0.0013291	0.0014921	0.0012827	0.0012045	0.0013342
18	0.0014145	0.0008464	0.0011580	0.0009735	0.0017004	0.0012186
19	0.0010205	0.0009987	0.0014612	0.0009428	0.0015440	0.0011934
20	0.0014671	0.0013097	0.0012559	0.0010788	0.0010766	0.0012376
21	0.0008886	0.0013101	0.0010438	0.0010466	0.0012256	0.0011029
22	0.0011262	0.0008901	0.0009725	0.0010467	0.0012539	0.0010579
23	0.0009936	0.0010997	0.0009188	0.0006976	0.0006997	0.0008819
24	0.0009183	0.0009386	0.0009411	0.0010240	0.0009907	0.0009625
25	0.0009445	0.0011248	0.0008921	0.0009118	0.0013875	0.0010521
26	0.0011213	0.0007000	0.0006991	0.0009372	0.0007577	0.0008431
27	0.0008124	0.0009128	0.0008670	0.0008650	0.0009150	0.0008744
28	0.0006998	0.0007625	0.0011471	0.0007633	0.0006991	0.0008144
29	0.0008341	0.0006994	0.0008923	0.0008601	0.0007612	0.0008094
30	0.0008385	0.0006998	0.0009385	0.0010688	0.0008378	0.0008767
31	0.0008082	0.0007321	0.0008622	0.0006997	0.0007886	0.0007782
32	0.0007580	0.0008891	0.0010998	0.0009114	0.0007876	0.0008892
33	0.0006991	0.0010956	0.0007869	0.0007832	0.0007000	0.0008130
34	0.0007859	0.0006990	0.0006996	0.0008887	0.0007000	0.0007546
35	0.0006999	0.0009130	0.0009437	0.0009905	0.0006985	0.0008491
36	0.0006980	0.0006969	0.0006999	0.0007824	0.0008360	0.0007426
37	0.0006996	0.0006998	0.0006994	0.0007866	0.0006954	0.0007162
38	0.0006982	0.0006998	0.0007567	0.0007858	0.0007596	0.0007400
39	0.0007578	0.0009434	0.0006980	0.0007000	0.0006958	0.0007590
40	0.0006999	0.0007867	0.0009152	0.0008638	0.0006979	0.0007927
41	0.0009369	0.0006978	0.0007877	0.0007860	0.0006982	0.0007813
42	0.0007573	0.0008597	0.0007000	0.0006999	0.0007821	0.0007598
43	0.0007556	0.0009428	0.0006976	0.0009917	0.0006988	0.0008173
44	0.0008644	0.0006998	0.0006935	0.0007579	0.0006993	0.0007430
45	0.0008619	0.0008368	0.0007574	0.0007868	0.0007844	0.0008055
46	0.0006966	0.0006989	0.0006985	0.0006902	0.0007808	0.0007130
47	0.0006968	0.0010186	0.0007851	0.0007603	0.0008355	0.0008193
48	0.0006995	0.0007830	0.0007868	0.0006939	0.0006994	0.0007325
49	0.0009386	0.0006979	0.0009410	0.0006987	0.0007844	0.0008121
50	0.0006985	0.0007863	0.0007559	0.0008648	0.0009123	0.0008036
51	0.0007860	0.0007562	0.0006997	0.0007573	0.0007847	0.0007568
52	0.0006992	0.0006987	0.0008594	0.0006997	0.0007582	0.0007430
53	0.0007590	0.0007580	0.0006990	0.0009137	0.0006998	0.0007659
54	0.0006998	0.0006997	0.0007565	0.0007849	0.0006998	0.0007281
55	0.0006998	0.0006988	0.0006924	0.0007561	0.0007821	0.0007258

Table F.8: All the Experiments Used to Find the Optimal Number of Hidden Neurons for the $BPNN_{11}$

Hidden Neurons	Run 1 MSE	Run 2 MSE	Run 3 MSE	Run 4 MSE	Run 5 MSE	Average MSE
14	0.0006998	0.0006900	0.0006604	0.0006072	0.0013870	0.0008089
15	0.0006996	0.0013865	0.0020807	0.0006997	0.0006997	0.0011132
16	0.0006996	0.0006998	0.0020783	0.0006982	0.0006999	0.0009752
17	0.0006997	0.0007000	0.0006268	0.0006999	0.0013862	0.0008225
18	0.0013863	0.0007000	0.0006999	0.0006998	0.0006088	0.0008190
19	0.0006996	0.0006998	0.0006561	0.0006998	0.0006121	0.0006735
20	0.0006995	0.0005853	0.0006898	0.0006733	0.0006997	0.0006695
21	0.0007000	0.0013869	0.0006993	0.0006998	0.0006996	0.0008371
22	0.0007000	0.0013859	0.0006999	0.0006999	0.0006829	0.0008337
23	0.0006998	0.0006728	0.0006998	0.0013872	0.0007000	0.0008319
24	0.0013865	0.0006997	0.0006996	0.0013865	0.0005875	0.0009520
25	0.0006998	0.0006202	0.0006997	0.0006984	0.0006998	0.0006836
26	0.0006602	0.0006999	0.0006804	0.0013870	0.0006215	0.0008098
27	0.0006997	0.0007000	0.0006978	0.0013866	0.0006868	0.0008342
28	0.0006865	0.0013879	0.0006996	0.0006999	0.0006109	0.0008170
29	0.0007000	0.0006877	0.0005753	0.0006635	0.0007000	0.0006653
30	0.0006999	0.0006998	0.0006837	0.0006995	0.0006999	0.0006966
31	0.0006921	0.0006998	0.0006992	0.0013869	0.0006680	0.0008292
32	0.0006998	0.0006996	0.0006998	0.0006997	0.0006877	0.0006973
33	0.0006496	0.0013870	0.0006998	0.0006384	0.0006997	0.0008149
34	0.0006998	0.0006967	0.0007000	0.0006999	0.0006999	0.0006993
35	0.0007000	0.0006852	0.0006304	0.0006833	0.0007000	0.0006798
36	0.0006236	0.0006233	0.0006945	0.0006996	0.0006998	0.0006682
37	0.0007000	0.0006346	0.0006001	0.0006999	0.0006999	0.0006669
38	0.0006641	0.0006966	0.0006091	0.0006909	0.0006998	0.0006721
39	0.0006130	0.0006994	0.0013872	0.0006999	0.0006998	0.0008199
40	0.0006998	0.0006996	0.0006998	0.0006999	0.0007000	0.0006998
41	0.0006998	0.0006607	0.0007000	0.0013875	0.0006997	0.0008295
42	0.0006950	0.0006998	0.0006998	0.0006826	0.0006999	0.0006954
43	0.0006608	0.0006932	0.0006017	0.0013863	0.0006997	0.0008083
44	0.0006486	0.0006995	0.0006995	0.0006595	0.0006999	0.0006814
45	0.0013869	0.0006958	0.0006450	0.0007000	0.0006996	0.0008255
46	0.0006232	0.0006999	0.0006997	0.0006503	0.0006754	0.0006697
47	0.0006390	0.0006729	0.0013870	0.0006987	0.0006861	0.0008167
48	0.0006879	0.0006435	0.0006434	0.0006996	0.0006997	0.0006748
49	0.0006999	0.0006997	0.0006999	0.0006898	0.0006997	0.0006978
50	0.0006999	0.0006999	0.0006997	0.0006641	0.0006996	0.0006926
51	0.0006999	0.0006581	0.0006536	0.0006999	0.0006972	0.0006817
52	0.0013870	0.0006921	0.0006996	0.0006997	0.0005771	0.0008111
53	0.0007000	0.0006005	0.0006916	0.0006568	0.0006999	0.0006698

Appendix G

Analysis of the Common Dictionary

This appendix contains some analysis of the common dictionary which was presented in Chapter 4. The analysis covers the following:

- Table G.1 : The frequency of each Arabic letter in the different root positions in the common dictionary.
- Table G.2 : The number of adjacency appearance of two letters in the positions 1 and 2 in the triliteral Roots in the common dictionary.
- Table G.3 : The number of adjacency appearance of two letters in the positions 1 and 3 in the triliteral Roots in the common dictionary.
- Table G.4 : The number of adjacency appearance of two letters in the positions 2 and 3 in the triliteral Roots in the common dictionary.

Letter	1st Position	2nd Position	3rd Position	Total
ر	395	503	488	1386
ن	453	391	426	1270
ل	336	429	454	1219
م	366	389	438	1193
ب	367	435	388	1190
و	417	559	200	1176
د	326	320	340	986
ف	312	320	326	958
ع	352	288	312	952
ق	324	272	303	899
س	322	253	293	868
ج	314	228	300	842
ح	326	250	247	823
ش	366	223	219	808
ك	313	224	235	772
ي	101	461	187	749
أ	290	205	251	746
ه	287	285	150	722
ط	212	211	259	682
ز	237	209	229	675
ت	194	244	227	665
خ	270	194	179	643
ص	220	154	180	554
غ	209	181	139	529
ث	155	171	163	489
ا	4	54	411	469
ض	180	135	111	426
ذ	133	170	111	414
ظ	50	58	79	187
ى	.	6	172	178
ء	.	11	18	29
إ	7	1	.	8
ئ	.	2	2	4
ؤ	.	1	.	1
آ	.	1	.	1
ة	.	.	.	0

Table G.1: The Frequency of Each Arabic Letter in the Different Root Positions in the Common Dictionary

Letter	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	ف	ق	ك	ل	م	ن	ه	و	ى	ي
ء	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
آ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
أ	.	.	.	.	.	.	.	24	.	11	10	9	5	9	13	7	21	16	15	13	8	6	6	2	1	1	12	7	9	15	16	14	7	17	.	16
ؤ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
إ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	4	1	.	.	.	.	1
ئ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ا	.	.	.	.	.	.	.	1	.	.	.	1	1	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ب	.	.	12	.	.	.	3	12	.	9	11	9	9	11	13	16	25	14	14	13	10	6	13	3	16	13	1	14	11	22	2	18	20	27	.	20
ة	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ت	.	.	9	.	.	.	3	10	.	4	3	5	5	6	.	.	20	1	6	3	.	.	2	.	8	6	10	6	6	18	9	13	3	22	.	16
ث	.	.	10	.	.	.	3	11	.	5	.	6	3	3	8	.	12	.	.	1	.	.	7	.	9	7	9	5	5	11	11	6	4	14	.	5
ج	.	.	15	.	.	.	.	20	.	4	7	3	13	9	14	9	25	13	9	9	1	2	3	2	14	1	14	1	1	24	18	21	14	28	.	20
ح	.	.	1	.	.	.	1	19	.	13	9	11	.	.	13	10	22	13	12	13	11	11	9	9	1	.	18	13	11	17	22	22	1	23	.	21
خ	.	.	.	.	.	.	.	17	.	12	7	8	.	.	12	11	19	13	12	14	9	11	9	2	5	.	15	3	.	17	16	21	.	23	.	14
د	3	.	12	.	.	.	2	19	.	.	8	10	17	10	9	1	22	5	12	7	2	1	1	1	19	13	14	13	12	20	20	17	16	24	1	15
ذ	4	.	11	.	.	.	1	8	.	.	.	2	6	2	.	1	14	.	.	1	.	.	1	.	10	2	4	5	5	9	8	2	7	15	.	15
ر	.	.	11	.	.	.	3	25	.	12	10	17	9	12	18	10	1	14	14	15	12	13	10	.	18	14	21	18	15	.	22	16	17	27	.	21
ز	.	.	12	.	.	.	5	17	.	3	.	8	8	7	10	1	17	3	.	.	.	.	1	.	14	10	9	12	9	13	12	16	11	21	.	18
س	.	.	10	.	.	.	5	21	.	13	.	14	15	14	20	4	19	.	4	.	.	.	11	.	11	7	20	14	10	21	18	19	13	20	.	19
ش	.	.	10	.	.	.	.	21	.	9	5	12	18	15	11	12	27	5	8	2	7	.	12	6	13	11	14	18	19	19	16	21	10	24	.	21
ص	1	.	6	.	.	.	1	13	.	6	.	1	10	9	12	1	14	.	.	.	2	.	7	.	12	6	13	9	3	17	15	16	9	19	.	18
ض	.	.	8	1	.	.	5	15	.	1	1	6	6	5	4	.	14	2	.	.	.	.	3	.	7	15	12	2	4	4	12	7	12	18	.	16
ط	.	.	1	.	.	.	1	16	.	1	4	1	14	8	1	.	18	3	13	3	.	.	2	.	10	8	12	2	.	19	15	14	14	20	.	12
ظ	.	.	5	.	.	1	1	5	.	.	.	1	.	.	.	.	6	.	.	1	.	.	.	.	1	.	2	.	.	5	4	4	2	7	.	5
ع	.	.	.	.	.	.	.	17	.	16	12	13	.	.	16	11	21	13	14	12	11	11	12	9	1	.	18	17	16	20	16	22	12	21	.	21
غ	.	.	.	.	.	.	2	14	.	4	6	1	.	.	10	7	16	7	10	10	3	10	9	2	.	.	8	1	.	19	17	14	3	13	5	18
ف	.	.	9	.	.	.	1	3	.	14	8	14	16	8	15	3	25	5	14	14	12	10	12	3	8	6	1	15	6	22	1	15	11	23	.	18
ق	.	.	5	.	.	.	1	17	.	10	9	.	14	5	12	9	26	9	11	13	10	8	13	.	18	.	16	7	.	23	18	18	11	24	.	17
ك	.	1	12	.	.	.	.	17	.	15	12	1	11	4	18	7	25	6	16	17	4	1	.	5	15	4	12	.	2	21	18	19	10	22	.	18
ل	1	.	7	.	.	.	2	23	.	12	12	12	17	14	8	9	3	10	8	4	9	4	13	4	17	12	14	17	18	1	21	2	15	27	.	20
م	2	.	13	.	.	1	2	1	.	19	7	10	18	12	14	12	25	12	16	14	11	11	13	2	18	15	1	11	11	24	3	14	15	19	.	20
ن	.	.	13	.	.	.	5	24	.	19	12	20	19	17	22	6	7	16	18	21	12	13	15	4	19	16	24	22	21	4	15	2	22	26	.	19
ه	.	.	1	.	.	.	1	22	.	12	3	15	.	1	18	9	23	12	7	4	5	7	9	.	2	2	7	13	10	18	20	19	1	23	.	23
و	.	.	11	.	1	.	5	16	.	16	13	17	15	13	20	13	25	13	13	16	14	9	14	4	17	12	15	21	18	18	20	12	21	2	.	13
ى	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ي	.	.	1	.	.	.	1	7	.	4	2	1	1	.	9	.	11	4	7	3	1	1	3	.	4	.	4	6	2	4	3	7	4	10	.	1

Table G.2: The Number of Sequential Appearance of Two Letters in Positions 1 and 2 in Triliteral Roots in the Common

Letter	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	ف	ق	ك	ل	م	ن	ه	و	ى	ي
ء	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
آ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
أ	.	.	10	.	.	.	18	13	.	13	4	10	8	7	15	2	19	7	9	4	5	6	6	2	2	2	14	12	13	18	15	23	10	6	8	9
ؤ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
إ	.	.	.	.	.	.	4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	1	1
ئ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ا	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	1	1	.	.	.
ب	.	.	11	.	.	.	21	7	.	15	11	15	11	8	13	3	26	12	18	16	8	7	13	5	17	13	1	18	13	25	20	16	7	6	5	6
ة	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ت	.	.	7	.	.	.	13	11	.	9	3	5	7	5	6	1	18	5	10	3	3	2	2	.	9	5	8	5	7	13	11	12	8	2	4	.
ث	.	.	5	.	.	.	10	12	.	7	2	9	2	4	10	.	11	.	.	2	.	.	9	.	7	5	3	4	1	13	10	11	2	4	7	5
ج	.	.	11	.	.	2	15	16	.	7	7	8	11	7	13	9	22	12	14	13	8	8	7	9	13	2	14	8	6	19	15	15	6	5	7	5
ح	.	.	16	.	.	.	21	16	.	8	9	11	4	.	16	7	21	10	10	10	10	7	10	6	.	.	16	11	12	24	18	19	1	17	4	12
خ	.	.	12	.	.	.	17	14	.	11	6	13	.	2	10	3	20	6	9	10	8	5	8	2	16	.	16	12	5	18	18	14	.	6	2	7
د	1	.	7	.	.	.	15	13	.	6	10	14	9	7	6	2	22	4	16	10	13	8	8	6	13	6	11	17	14	21	19	21	7	5	6	9
ذ	2	.	4	.	.	.	7	10	.	5	.	9	9	3	2	1	13	1	.	1	.	.	11	.	5	2	8	5	1	12	5	8	1	3	4	1
ر	2	.	15	.	.	.	19	20	.	6	8	12	16	14	20	5	1	14	18	13	9	8	11	1	18	11	19	17	14	20	25	23	8	11	8	9
ز	.	.	5	.	.	.	12	18	.	9	1	10	10	7	12	3	18	8	.	1	.	.	9	.	8	6	14	11	12	14	17	13	3	5	5	6
س	1	.	8	.	.	.	16	20	.	12	.	15	13	8	19	4	21	3	11	.	.	.	13	.	15	9	15	14	13	19	20	18	9	12	6	8
ش	1	.	11	.	.	.	22	20	.	9	10	10	14	9	9	11	23	13	11	8	11	1	11	6	15	7	18	12	9	21	19	21	8	14	7	5
ص	.	.	7	.	.	.	13	16	.	9	1	9	12	8	12	.	17	.	.	.	3	.	8	.	13	9	12	10	8	13	15	10	.	8	4	3
ض	.	.	7	.	.	.	14	8	.	3	2	8	5	3	9	.	11	14	8	.	.	2	10	.	10	3	6	3	9	10	9	14	1	6	2	3
ط	1	.	11	.	.	.	13	11	.	2	7	8	9	8	2	1	16	7	13	10	.	.	6	.	8	4	11	8	3	9	12	12	3	5	8	4
ظ	.	.	5	.	.	.	6	6	.	1	.	1	.	1	.	.	7	.	.	1	.	.	1	.	1	.	5	1	.	2	5	3	.	.	1	3
ع	1	.	2	.	.	.	20	20	.	7	10	15	.	.	20	4	21	12	15	14	11	5	16	3	3	.	17	18	14	22	22	22	13	13	3	9
غ	.	.	1	.	.	.	14	11	.	3	8	6	.	.	8	4	17	5	10	5	7	6	7	4	.	1	12	14	3	14	12	14	1	8	4	10
ف	.	.	15	.	.	.	17	3	.	8	4	15	17	13	15	6	20	10	12	10	8	6	7	3	14	9	2	13	9	21	16	12	6	11	4	6
ق	2	.	10	.	.	.	17	19	.	5	6	6	14	4	17	6	20	13	14	11	11	6	13	4	19	.	14	8	.	19	21	16	5	14	5	5
ك	.	.	11	.	.	.	14	18	.	13	8	12	14	6	16	5	20	9	13	10	10	1	6	6	13	1	15	1	10	16	17	15	9	6	9	9
ل	.	.	12	.	.	.	19	17	.	12	11	13	13	7	14	5	5	16	16	7	9	3	14	7	12	8	14	16	11	7	22	17	4	6	9	10
م	2	.	9	.	.	.	16	3	.	11	9	19	16	12	23	8	21	11	18	16	10	6	15	4	20	9	3	14	12	20	3	20	10	7	8	10
ن	2	.	14	.	.	.	17	22	.	16	11	18	17	13	15	8	25	14	18	20	16	11	17	3	19	9	22	16	14	25	23	6	11	13	9	9
ه	2	.	12	.	.	.	14	15	.	9	7	11	.	3	11	5	18	9	11	10	6	4	8	.	16	10	14	12	10	16	19	14	3	5	8	5
و	1	.	13	.	.	.	5	23	.	8	7	15	14	14	21	8	23	13	12	13	10	7	12	7	17	7	19	17	11	19	23	26	10	2	23	17
ى	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ي	.	.	.	.	.	.	2	6	.	3	1	3	2	6	6	.	12	1	7	1	3	2	1	1	9	1	3	6	1	2	7	10	3	.	1	1

Table G.3: The Number of Sequential Appearance of Two Letters in Positions 1 and 3 in Triliteral Roots in the Common

Letter	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	ف	ق	ك	ل	م	ن	ه	و	ى	ي
ء	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	1	.	1	1	1	.	.	.	.	.	.	.	.	.	2	2	.	.	.	.	.
آ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
أ	.	.	.	.	.	.	.	17	.	9	3	10	1	.	11	1	15	8	9	7	5	1	6	2	.	.	12	7	5	16	14	13	1	5	19	7
ؤ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.
إ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.
ئ	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ا	5	.	.	.	.	.	.	7	.	1	.	1	.	.	8	.	4	2	2	.	.	1	3	.	.	.	4	3	.	5	4	1	1	1	.	1
ب	1	.	23	.	.	.	21	27	.	15	16	21	14	13	13	13	25	12	20	16	13	6	17	1	16	10	.	19	14	22	5	24	12	8	9	9
ة	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ت	.	.	15	.	.	.	15	11	.	21	2	5	6	8	8	2	17	3	3	10	1	1	.	.	13	5	6	5	10	20	17	18	5	8	6	3
ث	.	.	10	.	.	.	14	5	.	2	19	8	2	2	6	.	16	.	.	.	.	.	8	.	9	6	4	5	1	14	14	13	1	3	3	6
ج	.	.	10	.	.	.	15	13	.	.	1	24	11	2	12	4	20	10	9	4	3	.	1	.	10	1	9	.	1	18	17	15	6	10	.	2
ح	.	.	.	.	.	.	16	12	.	10	9	10	21	.	9	1	15	9	9	7	9	6	11	5	.	.	14	7	7	16	16	16	.	9	3	3
خ	.	.	1	.	.	.	14	8	.	10	.	6	.	21	8	8	17	9	8	7	7	3	5	.	3	.	11	3	.	9	14	12	.	6	2	2
د	2	.	14	.	.	.	16	13	.	.	3	12	16	8	22	.	19	.	17	9	2	.	1	.	15	10	15	14	12	18	16	20	11	9	14	12
ذ	1	.	4	.	.	.	11	9	.	.	.	10	8	4	1	16	12	.	2	2	.	.	1	.	8	1	10	9	5	12	14	11	.	5	6	8
ر	4	.	17	.	.	.	20	24	.	17	15	23	16	10	23	5	25	22	21	18	14	16	20	2	19	9	20	24	18	11	21	18	13	11	12	15
ز	1	.	8	.	.	.	10	13	.	1	.	11	7	3	10	.	17	19	.	.	.	.	1	.	12	6	8	13	9	13	14	15	3	5	4	6
س	.	.	10	.	.	1	18	14	.	9	.	9	7	5	12	1	16	.	22	.	.	.	10	.	14	4	13	14	12	15	17	14	.	10	2	4
ش	.	.	9	.	.	1	14	14	.	5	1	5	9	3	7	1	17	3	1	21	1	.	9	3	9	5	8	11	9	16	13	12	.	6	5	5
ص	.	.	4	.	.	.	13	11	.	4	.	1	4	4	9	.	14	1	.	.	20	.	2	.	7	2	9	5	1	12	11	8	.	4	4	4
ض	.	.	5	.	.	.	11	8	.	.	.	5	5	6	7	1	8	2	1	.	.	17	2	1	7	3	6	.	2	8	10	6	1	6	3	4
ط	.	.	12	.	.	.	15	12	.	.	2	1	10	5	3	1	18	3	13	8	.	.	21	.	10	1	12	5	2	14	15	13	4	7	3	1
ظ	.	.	2	.	.	.	5	4	.	.	.	.	1	.	.	.	8	.	.	.	.	.	1	12	2	.	4	.	.	3	6	2	.	1	3	4
ع	.	.	.	.	.	.	16	17	.	8	8	10	.	.	12	1	20	11	13	6	10	8	15	8	23	.	14	10	6	23	15	15	1	10	3	5
غ	.	.	1	.	.	.	13	13	.	6	7	3	.	.	11	2	14	6	7	7	3	3	6	2	.	15	10	3	1	11	13	8	.	8	4	4
ف	.	.	12	.	.	.	17	.	.	14	8	10	11	8	14	3	21	10	13	9	8	6	9	4	13	6	22	15	6	21	3	21	10	11	6	9
ق	.	.	4	.	.	.	13	13	.	7	3	.	11	2	13	4	16	9	10	9	9	1	14	4	14	3	14	24	.	16	16	16	7	7	6	7
ك	.	.	11	.	.	.	12	12	.	7	6	.	6	1	13	.	16	7	9	6	5	3	1	3	11	.	10	.	22	15	16	13	4	6	4	5
ل	.	.	11	.	.	.	23	21	.	15	12	21	14	14	16	7	10	14	17	6	9	3	18	5	21	13	19	20	17	26	23	13	12	15	3	11
م	.	.	15	.	.	.	13	1	.	16	3	17	13	14	18	6	21	13	18	15	14	6	17	5	16	12	.	16	13	23	25	19	13	2	15	10
ن	.	.	14	.	.	.	18	22	.	13	9	21	15	11	15	8	15	11	16	13	9	2	13	8	15	5	19	15	16	3	21	26	6	9	12	11
ه	.	.	4	.	.	.	19	19	.	9	7	13	.	.	17	1	21	9	10	8	5	6	8	1	5	1	11	16	11	21	21	14	8	13	2	5
و	2	.	20	.	.	.	17	23	.	16	16	25	18	19	22	17	25	21	22	16	16	10	20	6	22	14	22	23	18	25	25	25	17	3	17	17
ى	.	.	.	.	.	.	.	1	.	.	1	.	.	.	1	.	1	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	.	.	.	.
ي	2	.	15	.	.	.	22	21	.	12	12	18	21	16	19	7	23	15	20	14	15	12	19	6	18	7	20	17	16	25	20	25	14	2	1	7

Table G.4: The Number of Sequential Appearance of Two Letters in Positions 2 and 3 in Triliteral Roots in the Common Dictionary